Spring JdbcTemplate

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 – jdbc template

Spring framework provides following approaches for JDBC database access:

- JdbcTemplate
- NamedParameterJdbcTemplate
- SimpleJdbcTemplate
- SimpleJdbcInsert and SimpleJdbcCall

Step 1:

created the following table inside the Oracle10g database.

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

Step 2:

Employee.java

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

```
package com.soften;

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.soften;
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.DriverManagerDataSource">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
<property name="username" value="bini" />
<property name="password" value="bini" />
</bean>

```

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

<bean id="edao" class="com.soften.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.soften;

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.xml");

    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);*/

}

}
```

PreparedStatement in Spring JdbcTemplate

We can execute parameterized query using Spring JdbcTemplate by the help of **execute()** method of JdbcTemplate class. To use parameterized query, we pass the instance of **PreparedStatementCallback** in the execute method.

Syntax of execute method to use parameterized query

```
public T execute(String sql,PreparedStatementCallback<T>);
```

PreparedStatementCallback interface

It processes the input parameters and output results. In such case, you don't need to care about single and double quotes.

Method of PreparedStatementCallback interface

It has only one method doInPreparedStatement. Syntax of the method is given below:

```
public T doInPreparedStatement(PreparedStatement ps)throws SQLException, DataAccessException
```

EmployeeDao.java

It contains one property jdbcTemplate and one method saveEmployeeByPreparedStatement. You must understand the concept of annonymous class to understand the code of the method.

```
package com.soften;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCallback;

public class EmployeeDao {
private JdbcTemplate jdbcTemplate;

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

public Boolean saveEmployeeByPreparedStatement(final Employee e){
   String query="insert into employee values(?,?,?)";
   return jdbcTemplate.execute(query,new PreparedStatementCallback<Boolean>(){
   @Override
   public Boolean doInPreparedStatement(PreparedStatement ps)
         throws SQLException, DataAccessException {

      ps.setInt(1,e.getId());
      ps.setString(2,e.getName());
      ps.setFloat(3,e.getSalary());

      return ps.execute();

   }
   });
}


}
```

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.DriverManagerDataSource">
<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.soften.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>


</beans>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the saveEmployeeByPreparedStatement() method.

```
package com.soften;

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.xml");

    EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
    dao.saveEmployeeByPreparedStatement(new Employee(108,"Amit",35000));
}
}
```

ResultSetExtractor Example | Fetching Records by Spring JdbcTemplate

We can easily fetch the records from the database using **query()** method of **JdbcTemplate** class where we need to pass the instance of ResultSetExtractor.

Syntax of query method using ResultSetExtractor

```
public T query(String sql,ResultSetExtractor<T> rse)
```

ResultSetExtractor Interface

ResultSetExtractor interface can be used to fetch records from the database. It accepts a ResultSet and returns the list.

Method of ResultSetExtractor interface

It defines only one method extractData that accepts ResultSet instance as a parameter. Syntax of the method is given below:

```
public T extractData(ResultSet rs)throws SQLException,DataAccessException
```

Example of ResultSetExtractor Interface to show all the records of the table

Employee.java

This class contains 3 properties with constructors and setter and getters. It defines one extra method toString().

```
package com.soften;

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

```java

public String toString(){
    return id+" "+name+" "+salary;
}
}
```

EmployeeDao.java

It contains on property jdbcTemplate and one method getAllEmployees.

```java
package com.soften;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.ResultSetExtractor;

public class EmployeeDao {
private JdbcTemplate template;

public void setTemplate(JdbcTemplate template) {
    this.template = template;
}

public List<Employee> getAllEmployees(){
 return template.query("select * from employee",new ResultSetExtractor<List<Employee>>(){
    @Override
     public List<Employee> extractData(ResultSet rs) throws SQLException,
            DataAccessException {

        List<Employee> list=new ArrayList<Employee>();
        while(rs.next()){
        Employee e=new Employee();
        e.setId(rs.getInt(1));
        e.setName(rs.getString(2));
        e.setSalary(rs.getInt(3));
        list.add(e);
        }
        return list;
        }
      });
    }
  }
```

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.DriverManagerDataSource">
<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.soften.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
```

```
</beans>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the getAllEmployees() method of EmployeeDao class.

```
package com.soften;

import java.util.List;

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.xml");
    EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
    List<Employee> list=dao.getAllEmployees();

    for(Employee e:list)
        System.out.println(e);

    }

}
```

RowMapper Example | Fetching records by Spring JdbcTemplate

Like ResultSetExtractor, we can use RowMapper interface to fetch the records from the database using **query()** method of **JdbcTemplate** class. In the execute of we need to pass the instance of RowMapper now.

Syntax of query method using RowMapper

```
public T query(String sql,RowMapper<T> rm)
```

RowMapper Interface

RowMapper interface allows to map a row of the relations with the instance of user-defined class. It iterates the ResultSet internally and adds it into the collection. So we don't need to write a lot of code to fetch the records as ResultSetExtractor.

Advantage of RowMapper over ResultSetExtractor

RowMapper saves a lot of code becuase it internally adds the data of ResultSet into the collection.

Method of RowMapper interface

It defines only one method mapRow that accepts ResultSet instance and int as the parameter list. Syntax of the method is given below:

```
public T mapRow(ResultSet rs, int rowNumber)throws SQLException
```

Example of RowMapper Interface to show all the records of the table

Employee.java

This class contains 3 properties with constructors and setter and getters and one extra method toString().

```
package com.soften;

public class Employee {
private int id;
private String name;
private float salary;
//no-arg and parameterized constructors
//getters and setters
public String toString(){
    return id+" "+name+" "+salary;
}
}
```

EmployeeDao.java

It contains on property jdbcTemplate and one method getAllEmployeesRowMapper.

```
package com.soften;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.ResultSetExtractor;
import org.springframework.jdbc.core.RowMapper;

public class EmployeeDao {
private JdbcTemplate template;

public void setTemplate(JdbcTemplate template) {
    this.template = template;
}

public List<Employee> getAllEmployeesRowMapper(){
 return template.query("select * from employee",new RowMapper<Employee>(){
    @Override
    public Employee mapRow(ResultSet rs, int rownumber) throws SQLException {
        Employee e=new Employee();
        e.setId(rs.getInt(1));
        e.setName(rs.getString(2));
        e.setSalary(rs.getInt(3));
        return e;
    }
    });
}
}
```

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.DriverManagerDataSource">
<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.soften.EmployeeDao">
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>

</beans>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the getAllEmployeesRowMapper() method of EmployeeDao class.

```
package com.soften;

import java.util.List;

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.xml");
```

```
        EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
        List<Employee> list=dao.getAllEmployeesRowMapper();

        for(Employee e:list)
            System.out.println(e);
    }
}
```