**@PathVariable**

*@PathVariable* is a Spring annotation which indicates that a method parameter should be bound to a URI template variable. If the method parameter is Map<String, String> then the map is populated with all path variable names and values.

It has the following optional elements:

* name - name of the path variable to bind to
* required - tells whether the path variable is required
* value - alias for name

The @ModelAttribute annotation is used as part of a Spring MVC web app and can be used in two scenarios.

* Firstly, it can be used to**inject data objects** in the model before a JSP loads. This makes it particularly useful by ensuring that a JSP has all the data it needs to display itself. The injection is achieved by binding a method return value to the model.
* Secondly, it can be used to**read data** from an existing model, assigning it to handler method parameters.

next →← prev

Spring JdbcTemplate Tutorial

Spring JDBC Template

Understanding the need for Spring JDBC Template

Advantage of Spring JDBC Template

JDBC Template classes

Example of JdbcTemplate class

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.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.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.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);\*/

}

[**next>>**](https://www.javatpoint.com/ResultSetExtractor-example)[**<<prev**](https://www.javatpoint.com/spring-JdbcTemplate-tutorial)

# Example of PreparedStatement in Spring JdbcTemplate

1. [PreparedStatement in Spring JDBC Template](https://www.javatpoint.com/example-of-PreparedStatement-in-Spring-JdbcTemplate)
2. [PreparedStatementCallback interface](https://www.javatpoint.com/example-of-PreparedStatement-in-Spring-JdbcTemplate)
3. [Example of using PreparedStatement in Spring](https://www.javatpoint.com/example-of-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

1. **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:

1. **public** T doInPreparedStatement(PreparedStatement ps)**throws** SQLException, DataAccessException

### Example of using PreparedStatement in Spring

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

1. create table employee(
2. id number(10),
3. name varchar2(100),
4. salary number(10)
5. );

**Employee.java**

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

1. **package** com.javatpoint;
3. **public** **class** Employee {
4. **private** **int** id;
5. **private** String name;
6. **private** **float** salary;
7. //no-arg and parameterized constructors
8. //getters and setters
9. }

**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.

1. **package** com.javatpoint;
2. **import** java.sql.PreparedStatement;
3. **import** java.sql.SQLException;
5. **import** org.springframework.dao.DataAccessException;
6. **import** org.springframework.jdbc.core.JdbcTemplate;
7. **import** org.springframework.jdbc.core.PreparedStatementCallback;
9. **public** **class** EmployeeDao {
10. **private** JdbcTemplate jdbcTemplate;
12. **public** **void** setJdbcTemplate(JdbcTemplate jdbcTemplate) {
13. **this**.jdbcTemplate = jdbcTemplate;
14. }
16. **public** Boolean saveEmployeeByPreparedStatement(**final** Employee e){
17. String query="insert into employee values(?,?,?)";
18. **return** jdbcTemplate.execute(query,**new** PreparedStatementCallback<Boolean>(){
19. @Override
20. **public** Boolean doInPreparedStatement(PreparedStatement ps)
21. **throws** SQLException, DataAccessException {
23. ps.setInt(1,e.getId());
24. ps.setString(2,e.getName());
25. ps.setFloat(3,e.getSalary());
27. **return** ps.execute();
29. }
30. });
31. }

34. }

**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.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="ds" **class**="org.springframework.jdbc.datasource.DriverManagerDataSource">
10. <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
11. <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
12. <property name="username" value="system" />
13. <property name="password" value="oracle" />
14. </bean>
16. <bean id="jdbcTemplate" **class**="org.springframework.jdbc.core.JdbcTemplate">
17. <property name="dataSource" ref="ds"></property>
18. </bean>
20. <bean id="edao" **class**="com.javatpoint.EmployeeDao">
21. <property name="jdbcTemplate" ref="jdbcTemplate"></property>
22. </bean>

25. </beans>

**Test.java**

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

1. **package** com.javatpoint;
3. **import** org.springframework.context.ApplicationContext;
4. **import** org.springframework.context.support.ClassPathXmlApplicationContext;
5. **public** **class** Test {
7. **public** **static** **void** main(String[] args) {
8. ApplicationContext ctx=**new** ClassPathXmlApplicationContext("applicationContext.xml");
10. EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
11. dao.saveEmployeeByPreparedStatement(**new** Employee(108,"Amit",35000));
12. }
13. }

}

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

1. **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:

1. **public** T extractData(ResultSet rs)**throws** SQLException,DataAccessException

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

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

1. create table employee(
2. id number(10),
3. name varchar2(100),
4. salary number(10)
5. );

**Employee.java**

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

1. **package** com.javatpoint;
3. **public** **class** Employee {
4. **private** **int** id;
5. **private** String name;
6. **private** **float** salary;
7. //no-arg and parameterized constructors
8. //getters and setters
10. **public** String toString(){
11. **return** id+" "+name+" "+salary;
12. }
13. }

**EmployeeDao.java**

It contains on property jdbcTemplate and one method getAllEmployees.

1. **package** com.javatpoint;
2. **import** java.sql.ResultSet;
3. **import** java.sql.SQLException;
4. **import** java.util.ArrayList;
5. **import** java.util.List;
6. **import** org.springframework.dao.DataAccessException;
7. **import** org.springframework.jdbc.core.JdbcTemplate;
8. **import** org.springframework.jdbc.core.ResultSetExtractor;
10. **public** **class** EmployeeDao {
11. **private** JdbcTemplate template;
13. **public** **void** setTemplate(JdbcTemplate template) {
14. **this**.template = template;
15. }
17. **public** List<Employee> getAllEmployees(){
18. **return** template.query("select \* from employee",**new** ResultSetExtractor<List<Employee>>(){
19. @Override
20. **public** List<Employee> extractData(ResultSet rs) **throws** SQLException,
21. DataAccessException {
23. List<Employee> list=**new** ArrayList<Employee>();
24. **while**(rs.next()){
25. Employee e=**new** Employee();
26. e.setId(rs.getInt(1));
27. e.setName(rs.getString(2));
28. e.setSalary(rs.getInt(3));
29. list.add(e);
30. }
31. **return** list;
32. }
33. });
34. }
35. }

**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.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="ds" **class**="org.springframework.jdbc.datasource.DriverManagerDataSource">
10. <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
11. <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
12. <property name="username" value="system" />
13. <property name="password" value="oracle" />
14. </bean>
16. <bean id="jdbcTemplate" **class**="org.springframework.jdbc.core.JdbcTemplate">
17. <property name="dataSource" ref="ds"></property>
18. </bean>
20. <bean id="edao" **class**="com.javatpoint.EmployeeDao">
21. <property name="jdbcTemplate" ref="jdbcTemplate"></property>
22. </bean>
24. </beans>

**Test.java**

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

1. **package** com.javatpoint;
3. **import** java.util.List;
5. **import** org.springframework.context.ApplicationContext;
6. **import** org.springframework.context.support.ClassPathXmlApplicationContext;
7. **public** **class** Test {
9. **public** **static** **void** main(String[] args) {
10. ApplicationContext ctx=**new** ClassPathXmlApplicationContext("applicationContext.xml");
11. EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
12. List<Employee> list=dao.getAllEmployees();
14. **for**(Employee e:list)
15. System.out.println(e);
17. }
19. }

# RowMapper Example | Fetching records by Spring JdbcTemplate

1. [RowMapper](https://www.javatpoint.com/RowMapper-example)
2. [RowMapper Interface](https://www.javatpoint.com/RowMapper-example)
3. [Method of RowMapper Interface](https://www.javatpoint.com/RowMapper-example)
4. [Example of RowMapper Interface](https://www.javatpoint.com/RowMapper-example)

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

1. **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:

1. **public** T mapRow(ResultSet rs, **int** rowNumber)**throws** SQLException

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

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

1. create table employee(
2. id number(10),
3. name varchar2(100),
4. salary number(10)
5. );

**Employee.java**

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

1. **package** com.javatpoint;
3. **public** **class** Employee {
4. **private** **int** id;
5. **private** String name;
6. **private** **float** salary;
7. //no-arg and parameterized constructors
8. //getters and setters
9. **public** String toString(){
10. **return** id+" "+name+" "+salary;
11. }
12. }

**EmployeeDao.java**

It contains on property jdbcTemplate and one method getAllEmployeesRowMapper.

1. **package** com.javatpoint;
2. **import** java.sql.ResultSet;
3. **import** java.sql.SQLException;
4. **import** java.util.ArrayList;
5. **import** java.util.List;
6. **import** org.springframework.dao.DataAccessException;
7. **import** org.springframework.jdbc.core.JdbcTemplate;
8. **import** org.springframework.jdbc.core.ResultSetExtractor;
9. **import** org.springframework.jdbc.core.RowMapper;
11. **public** **class** EmployeeDao {
12. **private** JdbcTemplate template;
14. **public** **void** setTemplate(JdbcTemplate template) {
15. **this**.template = template;
16. }
18. **public** List<Employee> getAllEmployeesRowMapper(){
19. **return** template.query("select \* from employee",**new** RowMapper<Employee>(){
20. @Override
21. **public** Employee mapRow(ResultSet rs, **int** rownumber) **throws** SQLException {
22. Employee e=**new** Employee();
23. e.setId(rs.getInt(1));
24. e.setName(rs.getString(2));
25. e.setSalary(rs.getInt(3));
26. **return** e;
27. }
28. });
29. }
30. }

**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.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3. xmlns="http://www.springframework.org/schema/beans"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
9. <bean id="ds" **class**="org.springframework.jdbc.datasource.DriverManagerDataSource">
10. <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
11. <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
12. <property name="username" value="system" />
13. <property name="password" value="oracle" />
14. </bean>
16. <bean id="jdbcTemplate" **class**="org.springframework.jdbc.core.JdbcTemplate">
17. <property name="dataSource" ref="ds"></property>
18. </bean>
20. <bean id="edao" **class**="com.javatpoint.EmployeeDao">
21. <property name="jdbcTemplate" ref="jdbcTemplate"></property>
22. </bean>
24. </beans>

**Test.java**

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

1. **package** com.javatpoint;
3. **import** java.util.List;
5. **import** org.springframework.context.ApplicationContext;
6. **import** org.springframework.context.support.ClassPathXmlApplicationContext;
7. **public** **class** Test {
8. **public** **static** **void** main(String[] args) {
9. ApplicationContext ctx=**new** ClassPathXmlApplicationContext("applicationContext.xml");
10. EmployeeDao dao=(EmployeeDao)ctx.getBean("edao");
11. List<Employee> list=dao.getAllEmployeesRowMapper();
13. **for**(Employee e:list)
14. System.out.println(e);
15. }
16. }