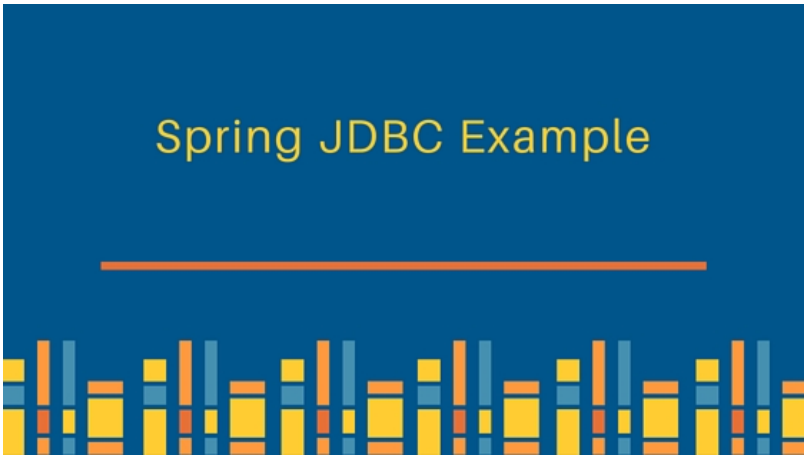
Spring JDBC Example

APRIL 2, 2018 BY PANKAJ — 63 COMMENTS

Spring JDBC is the topic of this tutorial. Databases are integral part of most of the Enterprise Applications. So when it comes to a Java EE framework, having good integration with JDBC is very important.

Table of Contents [hide]

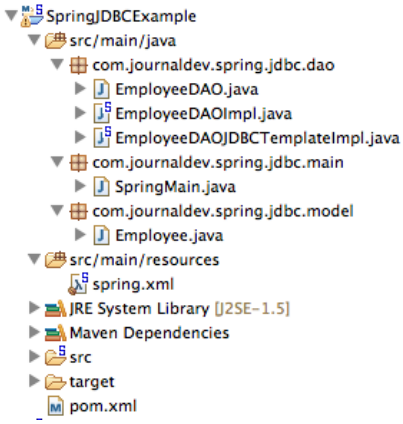
Spring JDBC



Spring Framework provides excellent integration with JDBC API and provides `JdbcTemplate` utility class that we can use to avoid bolier-plate code from our database operations logic such as Opening/Closing Connection, ResultSet, PreparedStatement etc.

Let's first look at a simple Spring JDBC example application and then we will see how JdbcTemplate class can help us in writing modular code with ease, without worrying whether resources are closed properly or not.

Spring Tool Suite to develop Spring based applications is very helpful, so we will use STS to create our Spring JDBC application. Our final project structure will look like below image.



Create a simple Spring Maven Project from the STS Menu, you can choose whatever name you like or stick with my project name as SpringJDBCExample.

Spring JDBC Dependencies

First of all we need to include Spring JDBC and Database drivers in the maven project pom.xml file. My final pom.xml file looks like below.

```
<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/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>org.springframework.samples</groupId>
        <artifactId>SpringJDBCExample</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <properties>

                <!-- Generic properties -->
                <java.version>1.6</java.version>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>

                <!-- Spring -->
                <spring-framework.version>4.0.2.RELEASE</spring-framework.version>
```

```
		<!-- Logging -->
```

Most of the part is automatically generated by STS, however I have update Spring Framework version to use latest version as 4.0.2.RELEASE.

Also we have added required artifacts **spring-jdbc** and **mysql-connector-java**. First one contains the Spring JDBC support classes and second one is database driver. I am using MySQL database for our testing purposes, so I have added MySQL JConnector jar dependencies. If you are using some other RDBMS then you should make the corresponding changes in the dependencies.

Spring JDBC Example – Database Setup

Let's create a simple table that we will use in our application for CRUD operations example.

```
CREATE TABLE `Employee` (
  `id` int(11) unsigned NOT NULL,
  `name` varchar(20) DEFAULT NULL,
  `role` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Spring JDBC Example – Model Class

We will use DAO Pattern for JDBC operations, so let's create a java bean that will model our Employee table.

```
package com.journaldev.spring.jdbc.model;

public class Employee {

	private int id;
	private String name;
	private String role;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getRole() {
		return role;
```

Spring JDBC Example – DAO Interface and Implementation

For DAO pattern, we will first have an interface declaring all the operations we want to implement.

```
package com.journaldev.spring.jdbc.dao;

import java.util.List;

import com.journaldev.spring.jdbc.model.Employee;

//CRUD operations
public interface EmployeeDAO {

	//Create
	public void save(Employee employee);
	//Read
	public Employee getById(int id);
	//Update
	public void update(Employee employee);
	//Delete
	public void deleteById(int id);
	//Get All
	public List<Employee> getAll();
}
```

```
package com.journaldev.spring.jdbc.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import javax.sql.DataSource;

import com.journaldev.spring.jdbc.model.Employee;

public class EmployeeDAOImpl implements EmployeeDAO {

	private DataSource dataSource;

	public void setDataSource(DataSource dataSource) {
		this.dataSource = dataSource;
	}

	@Override
```

The implementation of CRUD operations are simple to understand. If you want to learn more about DataSource, please read JDBC DataSource Example.

Spring JDBC Example – Bean Configuration

If you look at all the classes above, they are all using standard JDBC API and there is no reference to Spring JDBC framework. Spring JDBC framework classes comes into picture when we create Spring Bean

Configuration file and define the beans. We will create the DataSource in the Spring Bean context file and set it to our DAO implementation class.

My Spring Bean Configuration file looks like below.

```
<?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.xsd">

	<bean id="employeeDAO" class="com.journaldev.spring.jdbc.dao.EmployeeDAOImpl">
		<property name="dataSource" ref="dataSource" />
	</bean>

	<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/TestDB" />
		<property name="username" value="pankaj" />
		<property name="password" value="pankaj123" />
	</bean>

</beans>
```

First of all we are creating a DataSource object of class `DriverManagerDataSource`. This class provides the basic implementation of DataSource that we can use. We are passing MySQL database URL, username and password as properties to the DataSource bean.

Again dataSource bean is set to the `EmployeeDAOImpl` bean and we are ready with our Spring JDBC implementation. The implementation is loosely coupled and if we want to switch to some other implementation or move to other database server, all we need is to make corresponding changes in the bean configurations. This is one of the major advantage provided by Spring JDBC framework.

Spring JDBC Test Class

Let's write a simple test class to make sure everything is working fine.

```
package com.journaldev.spring.jdbc.main;

import java.util.List;
import java.util.Random;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.journaldev.spring.jdbc.dao.EmployeeDAO;
import com.journaldev.spring.jdbc.model.Employee;

public class SpringMain {
```

```
        public static void main(String[] args) {
                //Get the Spring Context
                ClassPathXmlApplicationContext ctx = new
ClassPathXmlApplicationContext("spring.xml");

                //Get the EmployeeDAO Bean
                EmployeeDAO employeeDAO = ctx.getBean("employeeDAO",
EmployeeDAO.class);
```

I am using Random Class to generate random number for employee id. When we run above program, we get following output.

```
Mar 25, 2014 12:54:18 PM
org.springframework.context.support.ClassPathXmlApplicationContext prepareRefresh
INFO: Refreshing
org.springframework.context.support.ClassPathXmlApplicationContext@4b9af9a9: startup
date [Tue Mar 25 12:54:18 PDT 2014]; root of context hierarchy
Mar 25, 2014 12:54:18 PM
org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [spring.xml]
Mar 25, 2014 12:54:19 PM org.springframework.jdbc.datasource.DriverManagerDataSource
setDriverClassName
INFO: Loaded JDBC driver: com.mysql.jdbc.Driver
Employee saved with id=726
Employee Found::{ID=726,Name=Pankaj,Role=Java Developer}
Employee Retrieved::{ID=726,Name=Pankaj,Role=Java Developer}
Employee updated with id=726
[{ID=726,Name=Pankaj,Role=CEO}]
Employee deleted with id=726
Mar 25, 2014 12:54:19 PM
org.springframework.context.support.ClassPathXmlApplicationContext doClose
INFO: Closing
org.springframework.context.support.ClassPathXmlApplicationContext@4b9af9a9: startup
date [Tue Mar 25 12:54:18 PDT 2014]; root of context hierarchy
```

Spring JdbcTemplate Example

If you look at the DAO implementation class, there is a lot of boiler-plate code where we are opening and closing Connection, PreparedStatements and ResultSet. This can lead to resource leak if someone forgets to close the resources properly. We can use `org.springframework.jdbc.core.JdbcTemplate` class to avoid these errors. Spring JdbcTemplate is the central class in Spring JDBC core package and provides a lot of methods to execute queries and automatically parse ResultSet to get the Object or list of Objects.

All we need is to provide the arguments as Object array and implement Callback interfaces such as `PreparedStatementSetter` and `RowMapper` for mapping arguments or converting ResultSet data to bean objects.

Let's look at another implementation of EmployeeDAO where we will use Spring JdbcTemplate class for executing different types of queries.

```
package com.journaldev.spring.jdbc.dao;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;

import com.journaldev.spring.jdbc.model.Employee;

public class EmployeeDAOJDBCTemplateImpl implements EmployeeDAO {

        private DataSource dataSource;

        public void setDataSource(DataSource dataSource) {
                this.dataSource = dataSource;
        }
```

Important points to look into above code for Spring JdbcTemplate are:

- Use of Object array to pass PreparedStatement arguments, we could also use PreparedStatementSetter implementation but passing Object array seems easy to use.
- No code related to opening and closing connections, statements or result set. All that is handled internally by Spring JdbcTemplate class.
- RowMapper anonymous class implementation to map the ResultSet data to Employee bean object in *queryForObject()* method.
- *queryForList()* method returns list of Map whereas Map contains the row data mapped with key as the column name and value from the database row matching the criteria.

To use Spring JdbcTemplate implementation, all we need is to change the employeeDAO class in the Spring Bean configuration file as shown below.

```
<bean id="employeeDAO"
class="com.journaldev.spring.jdbc.dao.EmployeeDAOJDBCTemplateImpl">
        <property name="dataSource" ref="dataSource" />
</bean>
```

When you will run the main class, the output of Spring JdbcTemplate implementation will be similar to the one seen above with normal JDBC implementation. That's all for Spring JDBC Example tutorial, download the sample project from below link and play around with it to learn more.

Download Spring JDBC Project

« PREVIOUS
Spring AOP Example Tutorial – Aspect, Advice, Pointcut, JoinPoint, Annotations, XML Configuration

NEXT »
Spring DataSource JNDI with Tomcat Example

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: SPRING

Comments

Meghana says
MARCH 26, 2018 AT 9:41 AM
Hi.,
Is it possible to use SpringJDBC to call a stored procedure that has TABLE TYPE as its IN and OUT Parameters?
I have tried "sqlArrayValues" and "SqlReturnArray()" i still get type mismatch.
I had to use simple JDBC to send TYPE TABLE parameters from a spring boot application.
Any help would be very much appreciated.
Reply

J Manas Kumar says
FEBRUARY 12, 2018 AT 5:30 AM
How can I fetch multiple records by single Id??
even if I write while loop instead of if…the result is same…can anyone help…???
Reply

J Manas Kumar says
JANUARY 31, 2018 AT 3:01 AM
public void delete(int empId) {
String sql = "delete from Employee where id=?";
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
int out = jdbcTemplate.update(sql,empId);
if(out !=0){
System.out.println("Employee deleted with id="+empId);
}else System.out.println("No Employee found with id="+empId);
}
in the above code I am facing issue at update(sql.,empId);
but in your code you just simply passed it .
its saying >>>>>> The method update(String, PreparedStatementSetter) in the type JdbcTemplate is not applicable for the arguments
(String, int)>>>>>>>>>>>
and another thing if my emp id is String how can I insert that into db??
and I m also facing this issue …..
Employee emp = jdbcTemplate.queryForObject(query, new Object[]{empId}, new RowMapper()
The type RowMapper is not generic; it cannot be parameterized with arguments
so can u suggest me anything???
m Waiting For your reply….
Thank you./..
Reply

Manas Naik says
JANUARY 13, 2018 AT 11:46 PM
Nice example … Grate efforts. □
I found 2 example for retrieve data from database.
1 – findAll() : Which return multiple record.
2 – findById() : Which return single value based on primary key as "Id".
Can you help me to resolve below query
How to retrieve data using jdbcTemplate based on name findByName() which return multiple record from data base?
Reply

Udit says
SEPTEMBER 8, 2017 AT 1:14 PM
Precise and Clear. Thanks.
Reply

Anupama says
FEBRUARY 22, 2017 AT 3:20 AM
Simple and straight forward explanation. Thanks Pankaj.
Reply

Colm says
NOVEMBER 15, 2016 AT 5:22 PM
Thanks for your efforts, much appreciated.
I'm using JDBCTemplate to do batch update, but PreparedStatement.setLong() throws a null pointer exception if the value is null:
preparedStatement.setLong(1, myBean.getLongVal()); // throws null pointer exception if myBean.getLongVal() = null
Is there a workaround for this? The corresponding database column is nullable, so its ok for it to have a null value. This is not a problem for string fields, i.e. the following is perfectly fine if the value is null:
preparedStatement.setLong(2, myBean.getStrVal()); // ok if myBean.getStrVal() = null
Reply

Praveen Kumar Natarajan says
OCTOBER 20, 2016 AT 11:36 AM
Hi , I'm trying to make it connect to my database(sybase) by joining the drivername,url ,username and password.but when i do so i get an exception like this
Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'employeeDAO' defined in class path resource [spring.xml]: Cannot resolve reference to bean 'dataSource' while setting bean property 'dataSource'; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'dataSource' defined in class path resource [spring.xml]: Error setting property values; nested exception is org.springframework.beans.PropertyBatchUpdateException; nested PropertyAccessExceptions (1) are:
PropertyAccessException 1: org.springframework.beans.MethodInvocationException: Property 'driverClassName' threw exception; nested exception is java.lang.IllegalStateException: Could not load JDBC driver class [com.sybase.jdbc3.jdbc.SybDriver]
at
org.springframework.beans.factory.support.BeanDefinitionValueResolver.resolveReference(BeanDefinitionValueResolver.java:328)
Reply

Pankaj says
OCTOBER 20, 2016 AT 8:45 PM
Looks like JDBC jar is missing from your project.
Reply

sandeep says

JANUARY 4, 2016 AT 11:40 PM

when i run programme with public class EmployeeDAOJDBCTemplateImpl {
}
Exception in thread "main" java.lang.NoClassDefFoundError: Could not initialize class org.springframework.jdbc.core.StatementCreatorUtils
please guys help me

Reply

Atif says

JANUARY 24, 2016 AT 3:58 PM

The version of your Spring core and JDBC should be the same.You can change that in pom.xml

Reply

Pankaj says

JUNE 6, 2016 AT 8:13 AM

Please check the versions and compare with my pom.xml code. These issue happens usually because of version mismatch and some class not compatible with other versions.

Reply

Sayali says

NOVEMBER 24, 2015 AT 1:20 AM

How we can call methods asynchronously to allow the queries to be performed in parallel. I would like to apply multi-threading concept here to load many-many users data parallel. How we can do that? Please help/suggest.
Thank you,
Sayali

Reply

Pankaj says

JUNE 6, 2016 AT 8:17 AM

Hi Sayali,
I have not tried this, so not sure. I would love to see our readers to pitch in for some solution.

Reply

vdep says

OCTOBER 21, 2015 AT 2:17 AM

Hi All,
I'am trying to query join operations between 2 model classes (lets say there is a manager class in addition to employee class), How does one do the below step:
while(rs.next()){
Employee emp = new Employee();
emp.setId(rs.getInt("id"));

emp.setName(rs.getString("name"));
emp.setRole(rs.getString("role"));
empList.add(emp);
}
I have class variables from both classes(lets say empid, managerid) in select part and class employee doesnt have managerid. so do i have to create a new class containing all the variables?
thanks
Reply

Pankaj says
JUNE 6, 2016 AT 8:25 AM
Look for ResultSetExtractor API, you need to implement it based on your requirements.
Reply

Ravi says
OCTOBER 19, 2015 AT 11:13 PM
Nice example for beginners. 100% clear !!!
Reply

Pankaj says
JUNE 6, 2016 AT 8:25 AM
Thanks Ravi.
Reply

deekonba says
OCTOBER 4, 2015 AT 6:55 PM
So far so good, Pankaj. Thanks for your example.
Do you have any spring's jdbctemplate example in spring4 with annotations and WITHOUT xml files but just only the coding?
Reply

Pankaj says
JUNE 6, 2016 AT 8:28 AM
I don't have it handy, but you can certainly convert XML based mapping to Annotation based mapping, check below tutorial for getting started.
https://www.journaldev.com/2461/spring-ioc-container-and-spring-bean-example-tutorial
Reply

Zac C says

SEPTEMBER 22, 2015 AT 4:48 PM

Is your Employee.sql db related to your spring.xml file where it says::

Reply

Zac C says

SEPTEMBER 23, 2015 AT 8:14 AM

Sorry. I don't know what happened, I meant to attach:

Reply

Pankaj says

JUNE 6, 2016 AT 8:30 AM

You can't attach a file, please paste the content here. Query??

Reply

Gerson Pereyra Garayar says

AUGUST 13, 2015 AT 6:49 AM

Hi, Pankaj

I'm from Peru and this tutotial is a great work but i have a mistake with the list of Products(instead of employee)

the others methods are OK (Save, Update, Delete and Get) but the method List all Products

in your example is the result: [[ID=726,Name=Pankaj,Role=CEO]]

but in my case is a project of 2 tutorials (this and Spring MVC Tutorial for Beginners with Spring Tool Suite) and the resulta was that:[com.tienda.spring.model.Producto@23d2a7e8]

Thanks

PD: Sorry if my english is bad

Reply

Pankaj says

JUNE 6, 2016 AT 8:31 AM

You need to override toString() method in your model bean.

Reply

Chandra says

APRIL 22, 2015 AT 7:21 AM

Good tutorial for all.

Reply

Pankaj says

JUNE 6, 2016 AT 8:31 AM

Thanks Chandra, appreciate it.

Reply

bhanu says
APRIL 21, 2015 AT 1:03 AM
I like tutorials with good terminology like your's....we can find bunch of examples in internet but less with proper java terminology
Reply

Pankaj says
JUNE 6, 2016 AT 8:35 AM
Thanks Bhanu for nice words.
Reply

Sanjay Jain says
FEBRUARY 24, 2015 AT 5:58 AM
Hi Nice article,
Just one query, In case of Spring Jdbc if id (primary key) is auto generated at database level, then How can I get that id in save api call?
Reply

Pankaj says
JUNE 6, 2016 AT 9:35 AM
You can use `SimpleJdbcInsert` class in this case, check `executeAndReturnKey` method.
Reply

Ferna says
FEBRUARY 17, 2015 AT 12:17 AM
thank you very much
Reply

Pankaj says
JUNE 6, 2016 AT 9:35 AM
you are welcome friend.
Reply

Rahul Gupta says
JANUARY 16, 2015 AT 3:02 AM

Hello Sir,
I always follow your tutorials. Thanks for sharing such a great knowledge.
While running above project, the values are not getting reflect into the database. I am using Oracle DB.
I made all the required changes. Its showing required output on console but not in database.!! Could you please solve my this query..??
I will truly appreciate your help..!!
Thanks in advance..!!
Reply

Pankaj says
JANUARY 16, 2015 AT 9:15 AM
I think auto commit is false in your case and you have missed the commit call.
Reply

oak says
JANUARY 6, 2015 AT 1:06 PM
thanks for the example, but could you explain how to do it with a join?
For example, if we have two tables: Employees and Salary and we want to join them on their id to get salararies for each employee, how should be the Model class?
It just has to have all the fields that we use in sql Select and all the rest logic remains the same?
Reply

Pankaj says
JUNE 6, 2016 AT 9:46 AM
Check ResultSetExtractor interface.
Reply

Monirul Islam says
DECEMBER 9, 2014 AT 2:40 AM
Hi,
Thank you for your tutorial. I am following your tutorial. But I am using mssql 2012 DB. Could you please tell me the changes i should made for MS SQL 2012 compability?
Reply

Evan says
DECEMBER 18, 2014 AT 9:47 PM
You can try to use the sql directly, because these sqls are very common and most possibly MS SQL 2012 use the same.
Reply

Pankaj says
JUNE 6, 2016 AT 9:47 AM

Thanks Evan, yes these should work just fine.
Reply

Kholofelo Maloma says
SEPTEMBER 18, 2014 AT 7:08 AM
Hi-
Thanks a lot for this post. Clear, loud and helpful. Thank you again I am new to Spring
Reply

Pankaj says
JUNE 6, 2016 AT 9:48 AM
you are welcome my friend.
Reply

Tony says
AUGUST 6, 2014 AT 1:31 AM
How can integrate it in a Spring Web Application? For example in my class called “Home Controller”:
*/
@RequestMapping(value = “/”, method = RequestMethod.GET)
public String home(Locale locale, Model model) {
//Get the Spring Context
ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext(“spring.xml”);
//Get the EmployeeDAO Bean
//EmployeeDAO employeeDAO = ctx.getBean(“employeeDAO”, EmployeeDAO.class);
//To use JdbcTemplate
EmployeeDAO employeeDAO = ctx.getBean(“employeeDAOJDBCTemplate”, EmployeeDAO.class);
//Run some tests for JDBC CRUD operations
Employee emp = new Employee();
int rand = new Random().nextInt(1000);
emp.setId(rand);
emp.setName(“Pankaj”);
emp.setRole(“Java Developer”);
//Create
employeeDAO.save(emp);
ctx.close();
return “home”;
}
This code return the follow error in Tomcat Server: Exception in thread “main”
java.lang.NoClassDefFoundError: Could not initialize class
org.springframework.jdbc.core.StatementCreatorUtils
at
org.springframework.jdbc.core.ArgumentPreparedStatementSetter.cleanupParameters(ArgumentPreparedStatementSetter.java:72)
at org.springframework.jdbc.core.JdbcTemplate$2.doInPreparedStatement(JdbcTemplate.java:924)

at org.springframework.jdbc.core.JdbcTemplate$2.doInPreparedStatement(JdbcTemplate.java:909)
at org.springframework.jdbc.core.JdbcTemplate.execute(JdbcTemplate.java:644)
at org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:909)
at org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:970)
at org.springframework.jdbc.core.JdbcTemplate.update(JdbcTemplate.java:980)
at
com.journaldev.spring.jdbc.dao.EmployeeDAOJDBCTemplateImpl.save(EmployeeDAOJDBCTemplateImpl.java:32)
Reply

TJ says
SEPTEMBER 3, 2014 AT 12:49 PM
Tony,
Did you find the solution? I have the same error
Reply

Francis says
FEBRUARY 2, 2015 AT 3:42 PM
I resolved the errors that Tony, TJ and myself saw when hitting the insert statements with parameters at runtime. Setting the same version as Pankaj did for both the framework and the jdbc worked for me, and here is what led me to that conclusion. I had been using some different versions since I was working on a different problem and reference material.
That is, the following basic error was thrown:
java.lang.NoClassDefFoundError: Could not initialize class
org.springframework.jdbc.core.StatementCreatorUtils
I saw that StatementCreatorUtils was an abstract class, with a static method and call, and since there is no "new" object of this class and no constructor, guessed there was just a release compatibility issue between the springframework-version and the springframework.jdbc-version so I tried a few Jar version combinations. When I just set the same versions as Pankaj had, apparently they were compatible and it worked. Also, I am working with MySql connector v. 5.1.30 (and MySql server itself is 5.6.17) as in below dependency:
4.0.2.RELEASE

4.0.2.RELEASE
.....
mysql
mysql-connector-java
5.1.30
compile
Reply

Murilo says
MARCH 22, 2015 AT 2:42 PM
Hey there!
BIG thanks from Brazil! ;D
Cheers!
Reply

praveen says
SEPTEMBER 22, 2015 AT 3:48 AM

good example

Atif says
JANUARY 24, 2016 AT 3:54 PM
Thanks.That solved the problem for me
Reply

Pankaj says
JUNE 6, 2016 AT 9:49 AM
Thanks Francis for pitching in and helping others. □
Reply

Evan says
DECEMBER 18, 2014 AT 9:50 PM
Try to add spring's @Transactional annotation to DAO impl class.
Reply

Francis says
JANUARY 29, 2015 AT 3:24 PM
Yes, any clues to Tony's question, Pankaj?? I am getting the same stack trace as Tony. Attempting to call a static method on the abstract StatementCreatorUtils class that is in the same library/jar.
I get the same whether I run in Eclipse/STS or maven spring-boot:run and it is clear the spring-jdbc dependency is defined otherwise the class doing the referencing would not be called since it is in the same jar.
package org.springframework.jdbc.core;
public abstract class StatementCreatorUtils {
public static void cleanupParameters(Object… paramValues) {
if (paramValues != null) {
cleanupParameters(Arrays.asList(paramValues));
}
}
Thanks.
Reply

Pankaj says
JUNE 6, 2016 AT 9:50 AM
Thanks for responding with fix, sorry for late response.
Reply

Kofi says
JULY 21, 2014 AT 11:31 AM
Friend thanks a lot for sharing, I have not taken the time to try because although it looks like a very good tutorial, the starting point in STS is always missing. Not enough to show the structure, STS has too many projects and it is hard to figure out which one to use for your tutorial. You could say for example "New-Spring-SpringMVC-create project.".
Reply

Pankaj says
JULY 21, 2014 AT 10:15 PM
You can easily figure it out. Its a standalone aplication, so obviously "Simple Spring Project"
Reply

Tricia says
JUNE 27, 2014 AT 6:04 AM
Where do you put the Employee.sql? How do you create the db for that?
Reply

Pankaj says
JULY 21, 2014 AT 10:16 PM
You need to create the database and run the acript for table generation. That part is out of acipe of this tutorial.
Reply

Zac C says
SEPTEMBER 22, 2015 AT 4:09 PM
I don't understand your reply? Are you saying we are to figure this out on our own? We need to have that database table talk to our spring project?
Reply

nithesh says
MAY 19, 2014 AT 3:54 AM
Cannot make a static reference to the non-static method save(Employee) from the type EmployeeInterface.....is the error which i m geting. coukld u plz help me to resolve this error???
Reply

Pankaj says
MAY 19, 2014 AT 8:44 AM
the error is clearly saying that you are trying to invoke non-static method from static method. Check your method signatures for save() and the method calling it.
Reply

akg says
MARCH 30, 2014 AT 6:38 PM
Great work on all your tutorials !
I am mainly a c++ guy trying to learn spring
However on this one I am getting several build errors in STS of type –
Multiple markers at this line
– The method save(Employee) of type EmployeeDAOJDBCTemplateImpl must override a superclass method
– implements com.journaldev.spring.jdbc.dao.EmployeeDAO.save
thanks a lot
Reply

akg says
MARCH 30, 2014 AT 7:30 PM
Found solution – it is java compiler setting in STS for the project required is >= 1.6 to override interface through annotation.
Reply

Manish says
APRIL 14, 2014 AT 1:24 AM
Same here .. I am also C++ guy but learning now Spring .. but it is quite easy to switch from C++ to Java.
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *
Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

SPRING FRAMEWORK

Spring Tutorial

Spring Core

- Spring Framework
- Spring Dependency Injection
- Spring IoC and Bean
- Spring Bean Life Cycle
- Spring REST
- Spring REST XML
- Spring RestTemplate
- Spring AOP
- Spring AOP Method Profiling
- Spring Annotations
- Spring @Autowired
- Spring @RequestMapping

Spring MVC

- Spring MVC Example
- Spring MVC Tutorial
- Spring MVC Exception Handling
- Spring MVC Validator
- Spring MVC Intercepter
- Spring MVC File Upload
- Spring MVC i18n
- Spring MVC Hibernate MqSQL

Spring ORM

- Spring ORM

RECOMMENDED TUTORIALS

- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial