Javarevisited
Blog about Java, Programming, Spring, Hibernate, Interview Questions, Books and Online Course Recommendations from Udemy, Pluarlsight etc

Home core java spring hibernate collections multithreading design patterns interview questions coding data structure OOP java 8 books About Me
Java Certifications JDBC jsp-servlet JSON SQL Linux Courses online resources jvm-internals REST Eclipse jQuery Java IO Java XML

WEDNESDAY, APRIL 5, 2017

Spring Framework Tutorial - How to call Stored Procedures from Java using IN and OUT parameter example

Spring Framework provides excellent support to call stored procedures from Java application. In fact there are multiple ways to call stored procedure in Spring Framework, e.g. you can use one of the `query()` method from `JdbcTemplate` to call stored procedures, or you can extend abstract class `StoredProcedure` to call stored procedures from Java. In this Java Spring tutorial, we will see second approach to call stored procedure. It's more object oriented, but same time requires more coding. `StoredProcedure` class allows you to declare IN and OUT parameters and call stored procedure using its various `execute()` method, which has protected access and can only be called from sub class. I personally prefer to implement `StoredProcedure` class as Inner class, if its tied up with one of DAO Object, e.g. in this case it nicely fit inside `EmployeeDAO`. Then you can provide convenient method to wrap stored procedure calls. In order to demonstrate, how to call stored procedures from spring based application, we will first create a simple stored proc using MySQL database, as shown below.

MySQL Stored procedure

We will use following stored procedure for this example. This is created in MySQL database and accept an input parameter IN, which is `employeeId` and return name of employee using its output parameter called, `name`.

```
mysql> DELIMITER //
mysql> create procedure usp_GetEmployeeName(IN id INT, OUT name VARCHAR(20))
    -> begin
    -> select emp_name into name from employee where emp_id = id;
    -> end//
Query OK, 0 rows affected (0.52 sec)

mysql> DELIMITER ;
```

For quick test, you can also call this stored procedure in mysql, assuming you have employee table as discussed in this article and some data on it. To learn more about stored proc in MySQL, see How to create and call MySQL stored procedure form command line.

```
mysql> call usp_GetEmployeeName(103, @name);
Query OK, 1 row affected (0.05 sec)

mysql> select @name;
+-------+
| @name |
+-------+
| Jack  |
+-------+
1 row in set (0.00 sec)
```

Spring Stored Procedure example and Configurations

Here is complete code example of how to call stored procedure from Spring framework. In this example, we have extended abstract class `StoredProcedure` in our class called, `EmployeeSP`. This is declared as nested class inside `EmployeeDAO` because its only used by this class, if your stored procedure is used my multiple DAO classes, than you can also make it a top level class. If you look at constructor of `EmployeeSP`, it calls super class constructor and passes datasource and name of database stored procedure. We have also declared two stored procedure parameters, one is IN parameter `id`, and other is OUT parameter. Input to stored procedure is passed using IN parameter, and output from stored procedure is read using OUT parameter. Your stored procedure can have multiple IN and OUT parameter. `StoredProcedure` class also provide several `execute()` methods, which can be invoked to call stored procedure and get result. It return result as Map, where key is OUT parameter, and value is result of stored procedure. Here is the code for DAO class and stored procedure along with Spring Configuration file, since Spring framework is based on principle of dependency Injection and Inversion of control, this file is required to create and manage object.

Java Class which wraps Stored procedure

```
import java.sql.Types;
import java.util.Map;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.SqlOutParameter;
import org.springframework.jdbc.core.SqlParameter;
import org.springframework.jdbc.object.StoredProcedure;

public class EmployeeDao {

        private JdbcTemplate jdbcTemplate;
        private EmployeeSP  sproc;

        public void setDataSource(DataSource source){
                this.jdbcTemplate = new JdbcTemplate(source);
                this.sproc = new EmployeeSP(jdbcTemplate.getDataSource());
        }

    /*
     * wraps stored procedure call
     */
        public String getEmployeeName(int emp_id){
                return (String) sproc.execute(emp_id);
        }

        /*
         * Inner class to implement stored procedure in spring.
         */
        private class EmployeeSP extends StoredProcedure{
                private static final String SPROC_NAME = "usp_GetEmployeeName";

                public EmployeeSP( DataSource datasource ){
                        super( datasource, SPROC_NAME );
                        declareParameter( new SqlParameter( "id", Types.INTEGER)

                        declareParameter( new SqlOutParameter( "name", Types.VARC
                        compile();
                }

                public Object execute(int emp_id){
                        Map<String,Object> results = super.execute(emp_id);
                        return results.get("name"); //reading output of stored pr
                }
        }
}
```

Main class to test stored procedure

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/*
 * Main class to start and test this Java application
 */
public class Main {

        public static void main(String args[]){
                ApplicationContext ctx = new ClassPathXmlApplicationContext("spri
                EmployeeDao dao = (EmployeeDao) ctx.getBean("employeeDao");

                //calling stored procedure using DAO method
                System.out.println("Employee name for id 103 is : " + dao.getEmpl
        }
}

Output
2013-01-17 23:56:34,408 0    [main] DEBUG EmployeeDao$EmployeeSP  - Compiled stor
2013-01-17 23:56:34,439 31   [main] DEBUG EmployeeDao$EmployeeSP  - RdbmsOperatic
Employee name for id 103 is : Jack
```

Spring configuration file:

```
<?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"
        xmlns:jms="http://www.springframework.org/schema/jms"
xmlns:context="http://www.springframework.org/schema/context"

        xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/jms
http://www.springframework.org/schema/jms/spring-jms-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

        <bean id="propertyPlaceholderConfigurer"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
                <property name="locations">
                        <list>
                                <value>classpath:jdbc.properties</value>
                        </list>
                </property>
        </bean>
```

```
    <bean id="springDataSource"
class="org.springframework.jdbc.datasource.SingleConnectionDataSource">
        <property name="driverClassName" value="${db.driver}" />
        <property name="url" value="${db.url}" />
        <property name="username" value="root" />
        <property name="password" value="root" />
    </bean>

        <bean id="employeeDao" class ="EmployeeDao">
                <property name="dataSource" ref="springDataSource"/>
        </bean>

</beans>
```

That's all on **How to call stored procedure from Java application using Spring Framework**. As discussed in 10 JDBC best practices for Java Programmer, JDBC API provides more straightforward approach using `CallableStatement`, but Spring's `StoredProcedure` class is also easy to use. You can also explore calling stored procedure, directly using JdbcTemplate in Spring.

Further Reading
Spring Framework 5: Beginner to Guru
Learn Spring Security by Eugen
Spring Master Class - Beginner to Expert
Introduction to Spring MVC 4 By Bryan Hansen
Spring and Hibernate for Beginners
Spring in Action 4th edition by Craig Walls

P.S. - If you want to learn how to develop RESTful Web Service using Spring MVC in depth, I suggest you join the REST with Spring certification class by Eugen Paraschiv. One of the best course to learn REST with Spring MVC.

By Javin Paul at April 05, 2017

Labels: core java , JDBC , spring
Location: United States