# [How to setup JNDI Database Connection pool in Tomcat - Spring Tutorial Example](http://javarevisited.blogspot.in/2013/01/how-to-setup-jndi-database-connection-pool-tomcat-spring-example-tutorial.html)

How to use JNDI database connection pool in Tomcat and Spring

There three *steps to configure and run JNDI Datasource Connection pool* for any  Java Web application:

1) Configure data source in Server and create JNDI name.

2) Configure [web.xml](http://javarevisited.blogspot.com/2011/12/load-on-startup-servlet-webxml-example.html)

3) Configure Spring bean with JNDI Datasource

4) Include [JDBC driver](http://javarevisited.blogspot.com/2012/05/different-types-of-jdbc-drivers-in-java.html) library on Server lib

In order to create JNDI DataSource on J2EE web server you need to follow server documentation. On Tomcat 6 you can simply put following piece of [XML](http://javarevisited.blogspot.com/2013/01/10-xml-interview-questions-and-answers.html) in context.xml to create Tomcat managed database connection pool:

**<?xml** version="1.0" encoding="UTF-8"**?>**  
**<Context** antiJARLocking="true" path="/springDataSourceDemo"**>**  
**<Resource** name="jdbc/springeDataSource"  
         auth="Container"  
         type="javax.sql.DataSource"  
         driverClassName="oracle.jdbc.driver.OracleDriver"  
         url="jdbc:oracle:thin:@localhost:1521:SPRING\_TEST"  
         username="root"  
         password="root"  
         removeAbandoned="true"  
         removeAbandonedTimeout="90"  
         logAbandoned="true"  
         maxActive="20"  
         maxIdle="10"  
         maxWait="-1"**/>**  
**</Context>**

Resource element will create JNDI data source which can be referenced using JNDI name "jdbc/springeDataSource". Tomcat internally use DBCP and Commons pool library for managing database connection pool. you can check tomcat/lib directory  for [jar file](http://javarevisited.blogspot.sg/2012/03/how-to-create-and-execute-jar-file-in.html) tomcat-dbcp.jar which is responsible for creating database connection pool inside [tomcat server](http://javarevisited.blogspot.com/2011/12/how-to-change-tomcat-default-port-8080.html).

**web.xml configuration to access JNDI Database connection pool**

In order to access any server resource from your web application, you need to specify JNDI resources in [web.xml](http://javarevisited.blogspot.com/2011/12/load-on-startup-servlet-webxml-example.html). you

can use the following XML to declare JNDI Datasource in web.xml:

**<resource-ref>**  
        **<description>**Oracle Spring JNDI Datasource**</description>**  
        **<res-ref-name>**jdbc/springDataSource**</res-ref-name>**  
        **<res-type>**javax.sql.DataSource**</res-type>**  
        **<res-auth>**Container**</res-auth>**  
**</resource-ref>**

Now your web application will see JNDI Datasource created in tomcat with name jdbc/springDataSource.

**Spring configuration for accessing JNDI Datasource :**

This spring configuration is generic enough which can be used to access any JNDI data source deployed on any J2EE Server. It’s not tied up with Tomcat. org.springframework.jndi.JndiObjectFactoryBean is used to lookup JNDI Datasource and bind with javax.sql.DataSource.

**<bean** id="springDataSource" class="org.springframework.jndi.JndiObjectFactoryBean"**>**  
  **<property** name="jndiName" value="java:comp/env/jdbc/springDataSource"**/>**  
  **<property** name="lookupOnStartup" value="true"**/>**  
  **<property** name="proxyInterface" value="javax.sql.DataSource"**/>**  
**</bean>**

Now the final step is to make sure tomcat lib has [JDBC driver jar file](http://javarevisited.blogspot.com/2012/03/jdbc-javalangclassnotfoundexception.html). I usually put JAR file inside lib directory of tomcat but you can put it anywhere it make sense and modify tomcat classpath to include driver JAR into [classpath](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html). Now rest of code which uses this datasource should remain same. You can get Spring DAO source from the previous article [How to setup Database Connection pool in Spring framework](http://javarevisited.blogspot.sg/2012/06/jdbc-database-connection-pool-in-spring.html).

# [JDBC Database connection pool in Spring Framework – How to Setup Example](http://javarevisited.blogspot.in/2012/06/jdbc-database-connection-pool-in-spring.html)

**Setting up JDBC Database Connection Pool in Spring framework** is easy for any Java application, just matter of changing few configuration in spring configuration file.If you are writing core java application and not running on any web or application server like Tomcat or  Weblogic,  Managing Database connection pool using **Apache Commons DBCP** and Commons Pool along-with Spring framework is nice choice but if you have luxury of having web server and managed J2EE Container, consider using **Connection pool managed by J2EE server** those are better option in terms of maintenance, flexibility and also help to prevent [java.lang.OutofMemroyError:PermGen Space in tomcat](http://javarevisited.blogspot.com/2012/01/tomcat-javalangoutofmemoryerror-permgen.html) by avoiding loading of JDBC driver in web-app class-loader, Also keeping JDBC connection pool information in Server makes it easy to change or include settings for JDBC over SSL. In this article we will see **how to setup Database connection pool in spring framework** using Apache commons DBCP and commons pool.jar

## Spring Example JDBC Database Connection Pool

**Spring framework** provides convenient JdbcTemplate class for performing all Database related operation. if you are not using Hibernate than using Spring's JdbcTemplate is good option. JdbcTemplate requires a DataSource which is javax.sql.DataSource implementation and you can get this directly using [spring bean](http://javarevisited.blogspot.sg/2012/05/what-is-bean-scope-in-spring-mvc.html) configuration or by using **JNDI** if you are using [J2EE web server or application server](http://javarevisited.blogspot.sg/2012/05/5-difference-between-application-server.html) for managing Connection Pool. See *How to setup JDBC connection Pool in tomcat and Spring* for JNDI based connection pooling for more details. In order to setup Data source you will require following configuration in your applicationContext.xml (spring configuration) file:

//Datasource connection settings in Spring  
**<bean** id="springDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close" **>**  
   **<property** name="url" value="jdbc:oracle:thin:@localhost:1521:SPRING\_TEST" **/>**  
   **<property** name="driverClassName" value="oracle.jdbc.driver.OracleDriver" **/>**  
   **<property** name="username" value="root" **/>**  
   **<property** name="password" value="root" **/>**  
   **<property** name="removeAbandoned" value="true"**/>**  
   **<property** name="initialSize" value="20" **/>**  
   **<property** name="maxActive" value="30" **/>**  
**</bean>**  
  
//Dao class configuration in spring  
 **<bean** id="EmployeeDatabaseBean" class="com.test.EmployeeDAOImpl"**>**  
    **<property** name="dataSource" ref="springDataSource"**/>**  
 **</bean>**

Below configuration of DBCP connection pool will create 20 database connection as initialSize is 20 and goes up to 30 Database connection if required as maxActive is 30. you can customize your database connection pool by using different properties provided by Apache DBCP library. Above example is creating connection pool with Oracle 11g database and we are using oracle.jdbc.driver.OracleDriver comes along with **ojdbc6.jar or ojdbc6\_g.jar ,**

## Java Code for using Connection pool in Spring

Below is **complete code example of DAO class which uses Spring JdbcTemplate** to execute SELECT query against database using database connection from Connection pool. If you are not initializing Database connection pool on start-up than it may take a while when you execute your first query because it needs to create certain number of SQL connection and then it execute query but once connection pool is created subsequent queries will execute faster.

*//Code for DAO Class using Spring JdbcTemplate*  
**package** com.test  
**import** javax.sql.DataSource;  
**import** org.log4j.Logger;  
**import** org.log4j.LoggerFactory;  
**import** org.springframework.jdbc.core.JdbcTemplate;  
  
***/\*\*  
 \* Java Program example to use DBCP connection pool with Spring framework  
 \* @author Javin Paul  
 \*/***  
**public** **class** EmployeeDAOImpl **implements** EmployeeDAO {  
  
    **private** **Logger** logger = LoggerFactory.getLogger(EmployeeDAOImpl.**class**);  
    **private** JdbcTemplate jdbcTemplate;  
  
    **public** **void** setDataSource([**DataSource**](http://java.sun.com/j2se/1.5.0/docs/api/javax/sql/DataSource.html) dataSource) {  
        **this**.jdbcTemplate = **new** JdbcTemplate(dataSource);  
    }  
  
    @**Override**  
    **public** **boolean** isEmployeeExists(**String** emp\_id) {  
        **try** {  
            logger.debug("Checking Employee in EMP table using Spring Jdbc Template");  
            **int** number = **this**.jdbcTemplate.queryForInt("select count(\*) from EMP where emp\_id=?", emp\_id);  
            **if** (number > 0) {  
                **return** **true**;  
            }  
        } **catch** (**Exception** exception) {  
            exception.printStackTrace();  
        }  
        **return** **false**;  
    }  
}

**Dependency:**

1. you need to include oracle driver jar like ojdbc\_6.jar in you classpath.

2. Apache DBCP and commons pool jar in application classpath.

That's all on **how to configure JDBC Database connection pool in Spring framework**. As I said its pretty easy using Apache DBCP library. Just matter of few configuration in spring applicationContext.xml and you are ready. If you want to configure *JDBC Connection pool on tomcat (JNDI connection pool)* and want to use in spring than see here.