Java DataSource, JDBC DataSource Example

APRIL 2, 2018 BY PANKAJ — 29 COMMENTS

Java DataSource and JDBC DataSource programming is the way to work with database in our java programs. We have already seen that JDBC DriverManager can be used to get relational database connections. But when it comes to actual programming, we want more than just connections.

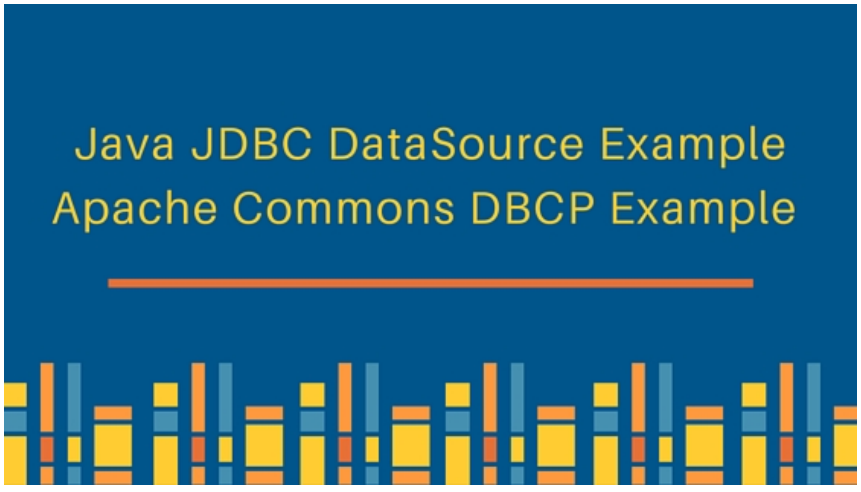


Table of Contents [hide]

Java DataSource

Most of the times we are looking for loose coupling for connectivity so that we can switch databases easily, connection pooling for transaction management and distributed systems support. **JDBC DataSource** is the preferred approach if you are looking for any of these features in your application. Java DataSource interface is present in `javax.sql` package and it only declare two overloaded methods `getConnection()` and `getConnection(String str1,String str2)`.

JDBC DataSource

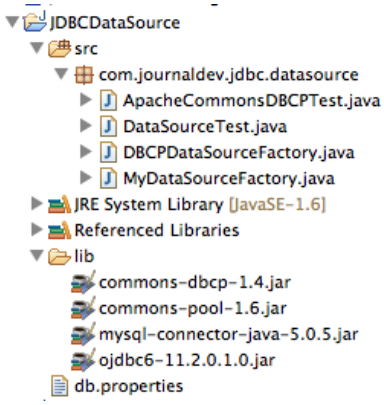
It is the responsibility of different Database vendors to provide different kinds of implementation of DataSource interface. For example MySQL JDBC Driver provides basic implementation of DataSource interface with `com.mysql.jdbc.jdbc2.optional.MysqlDataSource` class and Oracle database driver implements it with `oracle.jdbc.pool.OracleDataSource` class.

These implementation classes provide methods through which we can provide database server details with user credentials. Some of the other common features provided by these JDBC DataSource implementation classes are;

- Caching of PreparedStatement for faster processing
- Connection timeout settings
- Logging features
- ResultSet maximum size threshold

JDBC DataSource Example

Let's create a simple JDBC DataSource example project and learn how to use MySQL and Oracle DataSource basic implementation classes to get the database connection. Our final project will look like below image.



Java JDBC DataSource – Database Setup

Before we get into our example programs, we need some database setup with table and sample data. Installation of MySQL or Oracle database is out of scope of this tutorial, so I will just go ahead and setup table with sample data.

```
--Create Employee table
CREATE TABLE `Employee` (
```

```
  `empId` int(10) unsigned NOT NULL,
  `name` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`empId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- insert some sample data
INSERT INTO `Employee` (`empId`, `name`)
VALUES
        (1, 'Pankaj'),
        (2, 'David');

commit;



CREATE TABLE "EMPLOYEE"
  (
    "EMPID"   NUMBER NOT NULL ENABLE,
    "NAME"    VARCHAR2(10 BYTE) DEFAULT NULL,
    PRIMARY KEY ("EMPID")
  );

Insert into EMPLOYEE (EMPID,NAME) values (10,'Pankaj');
Insert into EMPLOYEE (EMPID,NAME) values (5,'Kumar');
Insert into EMPLOYEE (EMPID,NAME) values (1,'Pankaj');
commit;
```

Now let's move on to our java programs. For having database configuration loosely coupled, I will read them from property file.

db.properties file:

```
#mysql DB properties
MYSQL_DB_DRIVER_CLASS=com.mysql.jdbc.Driver
MYSQL_DB_URL=jdbc:mysql://localhost:3306/UserDB
MYSQL_DB_USERNAME=pankaj
MYSQL_DB_PASSWORD=pankaj123

#Oracle DB Properties
ORACLE_DB_DRIVER_CLASS=oracle.jdbc.driver.OracleDriver
ORACLE_DB_URL=jdbc:oracle:thin:@localhost:1521:orcl
ORACLE_DB_USERNAME=hr
ORACLE_DB_PASSWORD=oracle
```

Make sure that above configurations match with your local setup. Also make sure you have MySQL and Oracle DB JDBC jars included in the build path of the project.

Java JDBC DataSource – MySQL, Oracle Example

Let's write a factory class that we can use to get MySQL or Oracle DataSource.

```
package com.journaldev.jdbc.datasource;

import java.io.FileInputStream;
import java.io.IOException;
import java.sql.SQLException;
import java.util.Properties;

import javax.sql.DataSource;

import oracle.jdbc.pool.OracleDataSource;

import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;

public class MyDataSourceFactory {

	public static DataSource getMySQLDataSource() {
		Properties props = new Properties();
		FileInputStream fis = null;
		MysqlDataSource mysqlDS = null;
		try {
			fis = new FileInputStream("db.properties");
			props.load(fis);
```

Notice that both Oracle and MySQL DataSource implementation classes are very similar, let's write a simple test program to use these methods and run some test.

```
package com.journaldev.jdbc.datasource;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.sql.DataSource;

public class DataSourceTest {

	public static void main(String[] args) {

		testDataSource("mysql");
```

```
                System.out.println("**********");
                testDataSource("oracle");

        }

        private static void testDataSource(String dbType) {
                DataSource ds = null;
                if("mysql".equals(dbType)){
```

Notice that the client class is totally independent of any Database specific classes. This helps us in hiding the underlying implementation details from client program and achieve loose coupling and abstraction benefits.

When we run above test program, we will get below output.

```
Employee ID=1, Name=Pankaj
Employee ID=2, Name=David
**********
Employee ID=10, Name=Pankaj
Employee ID=5, Name=Kumar
Employee ID=1, Name=Pankaj
```

Apache Commons DBCP Example

If you look at above Java DataSource factory class, there are two major issues with it.

1. The factory class methods to create the MySQL and Oracle DataSource are tightly coupled with respective driver API. If we want to remove support for Oracle database in future or want to add some other database support, it will require code change.
2. Most of the code to get the MySQL and Oracle DataSource is similar, the only different is the implementation class that we are using.

Apache Commons DBCP API helps us in getting rid of these issues by providing Java DataSource implementation that works as an abstraction layer between our program and different JDBC drivers.

Apache DBCP library depends on Commons Pool library, so make sure they both are in the build path as shown in the image.

Here is the DataSource factory class using BasicDataSource that is the simple implementation of DataSource.

```
package com.journaldev.jdbc.datasource;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

import javax.sql.DataSource;
```

```
import org.apache.commons.dbcp.BasicDataSource;

public class DBCPDataSourceFactory {

        public static DataSource getDataSource(String dbType){
                Properties props = new Properties();
                FileInputStream fis = null;
                BasicDataSource ds = new BasicDataSource();

                try {
                        fis = new FileInputStream("db.properties");
                        props.load(fis);
                }catch(IOException e){
                        e.printStackTrace();
```

As you can see that depending on user input, either MySQL or Oracle DataSource is created. If you are supporting only one database in the application then you don't even need these logic. Just change the properties and you can switch from one database server to another. The key point through which Apache DBCP provide abstraction is *setDriverClassName()* method.

Here is the client program using above factory method to get different types of connection.

```
package com.journaldev.jdbc.datasource;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.sql.DataSource;

public class ApacheCommonsDBCPTest {

        public static void main(String[] args) {
                testDBCPDataSource("mysql");
                System.out.println("**********");
                testDBCPDataSource("oracle");
        }

        private static void testDBCPDataSource(String dbType) {
                DataSource ds = DBCPDataSourceFactory.getDataSource(dbType);

                Connection con = null;
                Statement stmt = null;
```

When you run above program, the output will be same as earlier program.

If you look at the Java JDBC DataSource and above usage, it can be done with normal DriverManager too. The major benefit of Java DataSource is when it's used within a Context and with JNDI.

With simple configurations we can create a Database Connection Pool that is maintained by the Container itself. Most of the servlet containers such as Tomcat and JBoss provide it's own Java DataSource implementation and all we need is to configure it through simple XML based configurations and then use JNDI context lookup to get the Java DataSource and work with it. This helps us by taking care of connection pooling and management from our application side to server side and thus giving us more time to write business logic for the application.

In next tutorial, we will learn how we can configure DataSource in Tomcat Container and use it in Web Application.

« PREVIOUS
CallableStatement in Java Example

NEXT »
Tomcat DataSource JNDI Example in Java

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: DATABASE, JAVA

Comments

Daniel says
MAY 7, 2018 AT 6:11 AM
Thanks.. This article is what I was looking for!
Reply

HANGNKM says
OCTOBER 27, 2017 AT 1:48 AM
where will db.properties be placed? please show me the structure of this project
Reply

Pankaj says
OCTOBER 27, 2017 AT 5:18 PM
Check the image, it's on Eclipse project root directory.
Reply

Hiten says
DECEMBER 19, 2016 AT 4:47 AM
A simple way to use the DatasourceFactory is to be use the below code
/* Invoke the database */
InitialContext jndiEnc = new InitialContext();
DataSource ds = (DataSource) jndiEnc.lookup("java:comp/env/jdbc/DBASE");
conn = ds.getConnection();
/* The Connection object received is an instance of PooledConnection.
Following line converts PooledConnection to OracleConnection. */
oracleconn = (OracleConnection) (Connection) ((PooledConnection) conn).getConnection();
You can find the detailed configuration to be done in the following post.
http://tenthsense.blogspot.in/2016/12/improve-jdbc-oracle-database.html
Reply

bellis says
JULY 3, 2017 AT 9:11 AM
Thank you!
Reply

Ath J says
AUGUST 9, 2016 AT 6:39 AM
Good tutorial ,But is connection pooling being used in the above examples ??
Reply

dave Johnson says
FEBRUARY 20, 2017 AT 10:41 AM
I also want to know the answer for this question. Pankaj please give the answer. Thankss!
I know that if you use dbcp's BasicDataSource instead of OracleDataSource or MysqlDataSource, you are getting connection pool already without further ado.
Reply

Pankaj says
FEBRUARY 21, 2017 AT 12:13 AM
In above examples we are creating direct connection and not using connection pool.
Reply

Ashutosh Kumar Dwivedi says
JULY 15, 2016 AT 9:58 PM
Getting this error ....
java.sql.SQLException: Access denied for user ''@'localhost' (using password: YES)
at com.mysql.jdbc.SQLError.createSQLException(SQLError.java:996)
at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3887)
at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3823)
at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:870)
at com.mysql.jdbc.MysqlIO.proceedHandshakeWithPluggableAuthentication(MysqlIO.java:1659)
at com.mysql.jdbc.MysqlIO.doHandshake(MysqlIO.java:1206)
at com.mysql.jdbc.ConnectionImpl.coreConnect(ConnectionImpl.java:2234)
at com.mysql.jdbc.ConnectionImpl.connectOneTryOnly(ConnectionImpl.java:2265)
at com.mysql.jdbc.ConnectionImpl.createNewIO(ConnectionImpl.java:2064)
at com.mysql.jdbc.ConnectionImpl.(ConnectionImpl.java:790)
at com.mysql.jdbc.JDBC4Connection.(JDBC4Connection.java:44)
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at sun.reflect.NativeConstructorAccessorImpl.newInstance(Unknown Source)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(Unknown Source)
at java.lang.reflect.Constructor.newInstance(Unknown Source)
at com.mysql.jdbc.Util.handleNewInstance(Util.java:377)
at com.mysql.jdbc.ConnectionImpl.getInstance(ConnectionImpl.java:395)
at com.mysql.jdbc.NonRegisteringDriver.connect(NonRegisteringDriver.java:325)
at com.mysql.jdbc.jdbc2.optional.MysqlDataSource.getConnection(MysqlDataSource.java:422)
at com.mysql.jdbc.jdbc2.optional.MysqlDataSource.getConnection(MysqlDataSource.java:134)
at com.mysql.jdbc.jdbc2.optional.MysqlDataSource.getConnection(MysqlDataSource.java:105)

at com.ashu.dbcp.TestDataSource.testDataSource(TestDataSource.java:28)
at com.ashu.dbcp.TestDataSource.main(TestDataSource.java:71)
Reply

Pankaj says
JULY 15, 2016 AT 10:40 PM
user name is empty, please check. Also check if you are able to connect using mysql command line.
Reply

Dan says
JUNE 24, 2016 AT 5:13 AM
Great tutorial! Thanks!
Reply

venkat says
MARCH 4, 2016 AT 10:44 PM
awesomw pankaj.... □
Reply

Bapan Dhali says
FEBRUARY 16, 2016 AT 9:02 PM
Could you please tell me why I am getting that type of exception when I run the program. Exception given below.
package com.java.test;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DatabaseTest {
public static void main(String[] args) {
try {
Class.forName("oracle.jdbc.OracleDriver");
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@//localhost:1521:orcle","system","Infy1234");
System.out.println("connected.."+con);
} catch (ClassNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (SQLException e) {
// TODO Auto-generated catch block
e.printStackTrace();

}
}
}
==========
Exception:
java.sql.SQLException: Io exception: The Network Adapter could not establish the connection
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java:112)
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java:146)
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java:255)
at oracle.jdbc.driver.T4CConnection.logon(T4CConnection.java:387)
at oracle.jdbc.driver.PhysicalConnection.(PhysicalConnection.java:414)
at oracle.jdbc.driver.T4CConnection.(T4CConnection.java:165)
at oracle.jdbc.driver.T4CDriverExtension.getConnection(T4CDriverExtension.java:35)
at oracle.jdbc.driver.OracleDriver.connect(OracleDriver.java:801)
at java.sql.DriverManager.getConnection(Unknown Source)
at java.sql.DriverManager.getConnection(Unknown Source)
at com.infy.DatabaseTest.main(DatabaseTest.java:11)
Reply

Pankaj says
JUNE 11, 2016 AT 12:02 PM
Seems like your Database connection is not working properly. Please check user, password settings.
Reply

Shifar says
DECEMBER 17, 2015 AT 6:51 AM
Listen please,
I've read this article and an obviously it's working perfectly. but i've a doubt about Connection pooling.
What is the difference between this method and the method available in this link :
https://www.mulesoft.com/tcat/tomcat-mysql ? Which one is better method ?
Reply

Pankaj says
JUNE 11, 2016 AT 12:03 PM
Best option is to use server for creating connection pool and then use it in our application by getting connection from pool using JNDI.
Reply

DK says
JUNE 21, 2016 AT 6:54 AM
I am looking at exackly that, can you please explain how I can do that. I have created a datasource in Jboss AS 7 and have given it a JNDI name java:/myOwndatasource. How do lookup for this datasource in my application.

Reply

mario says
AUGUST 20, 2015 AT 8:31 PM
thanks!
Reply

Ravi Kant Gaur says
JULY 14, 2015 AT 12:17 PM
Hi
I am using apache common dbcp 1.2.2 jar to create the database datasource in my project where database side we have upgraded the database Oracle 11.2.0.4.1 to Oracle 11.2.0.4.5.
below is the my code:
where
1. driver=oracle.jdbc.driver.OracleDriver
2. url=jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=oramtbdocq.qs2x.vwg)(PORT=1560)))(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=mtbdocq.qs2x.vwg)))
3. initialsize=10
4. maxidle=10
5. maxactive=10
we are using ojdbc7.jar since my jdk is 1.7
Code 1: *Trying to make jdbc connection by using dbcp 1.2.2 jar*
try{
org.apache.commons.dbcp.BasicDataSource dataSource = new BasicDataSource();
dataSource.setDriverClassName(driver);
dataSource.setUrl(url);
dataSource.setUsername(username);
dataSource.setPassword(password);
dataSource.setInitialSize(initialsize);
dataSource.setMaxIdle(maxidle);
dataSource.setMaxActive(maxactive);
basicDataSourceCon =dataSource.getConnection();
System.out.println("\n basicDataSourceCon ::"+basicDataSourceCon);
System.out.println("\n\n<<<<<>>>>>\n\n");
}catch(Exception e2){
System.out.println("\n <<<<<<<<<<>>>>>>>>>>> ::");
e2.printStackTrace(); }
Exception:
at org.apache.commons.dbcp.BasicDataSource.createDataSource(BasicDataSource.java:1225)
at org.apache.commons.dbcp.BasicDataSource.getConnection(BasicDataSource.java:880)
at

com.emc.xcp.bam.bamserver.listener.BamContextLoaderListener.jdbctestConnection(BamContextLoaderListener.java:265)
at
com.emc.xcp.bam.bamserver.listener.BamContextLoaderListener.contextInitialized(BamContextLoaderListener.java:92)
at org.apache.catalina.core.StandardContext.listenerStart(StandardContext.java:4797)
at org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5291)
at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:150)
at org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase.java:901)
at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:877)
at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:633)
at org.apache.catalina.startup.HostConfig.deployDirectory(HostConfig.java:1114)
at org.apache.catalina.startup.HostConfig$DeployDirectory.run(HostConfig.java:1673)
at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:471)
at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:334)
at java.util.concurrent.FutureTask.run(FutureTask.java:166)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
at java.lang.Thread.run(Thread.java:722)
Caused by: java.sql.SQLException: Io exception: Oracle Error ORA-12650
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java:113)
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java:147)
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java:257)
at oracle.jdbc.driver.T4CConnection.logon(T4CConnection.java:389)
at oracle.jdbc.driver.PhysicalConnection.(PhysicalConnection.java:454)
at oracle.jdbc.driver.T4CConnection.(T4CConnection.java:165)
at oracle.jdbc.driver.T4CDriverExtension.getConnection(T4CDriverExtension.java:35)
at oracle.jdbc.driver.OracleDriver.connect(OracleDriver.java:802)
at
org.apache.commons.dbcp.DriverConnectionFactory.createConnection(DriverConnectionFactory.java:38)
at
org.apache.commons.dbcp.PoolableConnectionFactory.makeObject(PoolableConnectionFactory.java:294)
at org.apache.commons.dbcp.BasicDataSource.validateConnectionFactory(BasicDataSource.java:1247)
at org.apache.commons.dbcp.BasicDataSource.createDataSource(BasicDataSource.java:1221)
… 17 more
But when I am using the simple jdbc connection it is going fine.
Code 2: *Simple jdbc connection which is going fine.*

```
try{
Properties props=new Properties();
props.put("user", username);
props.put("password", password);
props.put("maxIdle", maxidle);
props.put("maxActive", maxactive);
props.put("initialSize", initialsize);
Class.forName(driver);
DriverManager.getDriver(url);
simplecon = DriverManager.getConnection(url,props);
//DriverManager.getConnection(url,username,password);
System.out.println("\n Simple JDBC simplecon ::"+simplecon);
System.out.println("\n\n<<<<<<>>>>>\n\n");
}catch(Exception e1){
```

System.out.println("\n <<<<<<<<<<>>>>>>>>>>>>>>> ::");
e1.printStackTrace();
}
Any Idea? What could be the root cause.
Thanks in advance.
Reply

Pankaj says
JUNE 11, 2016 AT 12:06 PM
I am not sure what is wrong by looking at your code snippet, if you can share that with me then I might have a look at it.
Reply

LingaMoorthy Reddy says
APRIL 9, 2015 AT 11:48 PM
Hello Sir,
I want crate a table in oracle using remote desktop connection then I want write java code for import that data from oracle to mysql in my system. Please help me to write a code for this issue. Really I don't know how to write a code for it. Kindly, request you to help me in this code.
Thnaks
N LingaMoorthy Reddy
Reply

Pankaj says
JUNE 11, 2016 AT 12:07 PM
I have not worked on anything like this, please use Google. I am sure you will find something to get started.
Reply

marcos says
MARCH 18, 2015 AT 12:22 AM
Can this talk to any database by just adding more jars?
Reply

Pankaj says
JUNE 11, 2016 AT 12:07 PM
Depends if the database is supported, you can check it out.
Reply

sumit says
JANUARY 4, 2015 AT 10:38 PM
Why have you defined- ORACLE_DB_DRIVER_CLASS=oracle.jdbc.driver.OracleDriver in props file. We are not using it anywhere in our code?
Reply

Pankaj says
JUNE 11, 2016 AT 12:08 PM
This is to keep it configurable. We are using it to work with Oracle database using Java JDBC DataSource.
Reply

jonnyhaski says
DECEMBER 4, 2014 AT 1:07 AM
Hello.
I am setting up Oracle SQLdeveloper to work with mySQL. I've setted up driver and login details. But have error while connecting to database
An error was encountered performing the requested operation:
Packet for query is too large (4739923 > 1048576). You can change this value on the server by setting the max_allowed_packet' variable.
Vendor code 0
Is there any suggestions how to fix this?
I don't have mysql server on client machine(windows 7). Only driver and sqldeveloper installed. And I don't want to reduce max_allowed_packet size on my debian server. I've read this doc http://www.ccs.neu.edu/home/kathleen/classes/cs3200/connector-j.pdf But couldn't understand how to edit this http://clip2net.com/s/jpjkq6
Thank you in advance.
Reply

Sumit Chouksey says
JUNE 15, 2014 AT 11:31 PM
can u please give the JDBC program for Disconnected Architecture which uses the javax.sql package
Reply

Siddu says
JANUARY 25, 2014 AT 3:41 AM
Could you post RowSet interface concepts.
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

CORE JAVA TUTORIAL

Java 10 Tutorials Java 9 Tutorials Java 8 Tutorials Java 7 Tutorials Core Java Basics OOPS Concepts Data Types and Operators String Manipulation Java Arrays Annotation and Enum Java Collections Java IO Operations Java Exception Handling MultiThreading and Concurrency Regular Expressions Advanced Java Concepts

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- 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