JDBC Connection Pooling

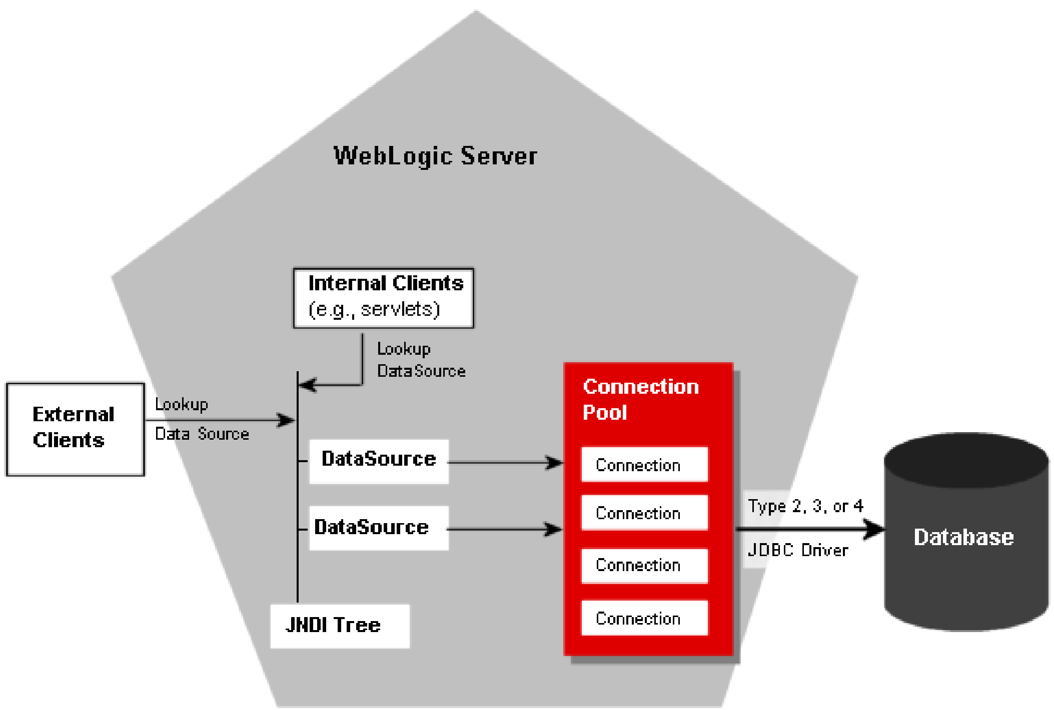
6.0 Connection Pooling

When an application calls the DataSource.getConnection method, the return value is always a Connection object. As I already explained the Connection object can vary depending on the implementation of the DataSource class.

6.1 How Connection Pool works?

When the DataSource class implements connection pooling, the Connection object that the method getConnection returns to the application is actually a handle to a PooledConnection object. This handle delegates most of the work of the connection to the underlying PooledConnection object, which is a physical connection, so the application is never aware that its Connection object is not a regular physical connection.

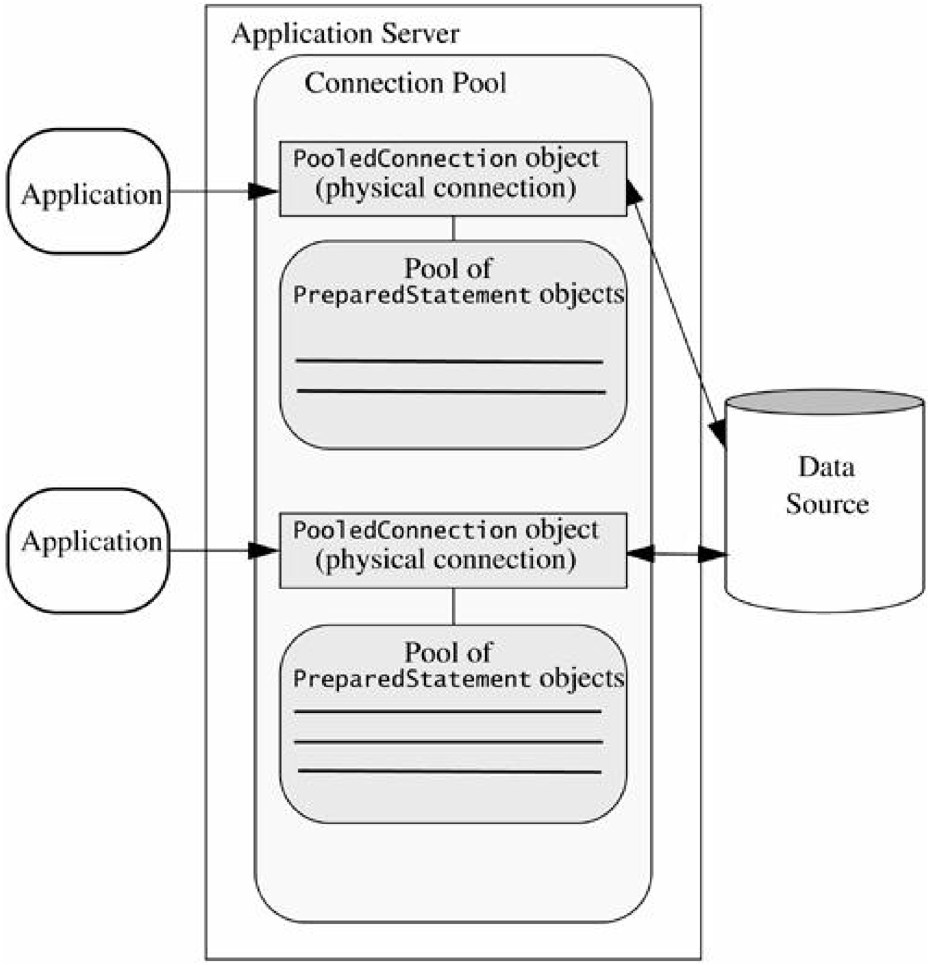
In this case, when the method getConnection is invoked, connection pooling checks if there is a PooledConnection object available for resuse in the pool of connections. If there is, it will be resued. if there is no reusable PooledConnection object, a ConnectinPoolDataSource object will be used to create PooledConnection. In either case, a reusable PooledConnection object will have the method pooledConnection.getConnection called on it to create the Connection object to return to the application.



6.2 Statement Pooling

With the new specification, a PooledConnection object can have a pool of PreparedStatement objects associated with it. This allows the reuse of a PreparedStatement object similar to the way a pool of PooledConnection objects allows the reuse of a connection.

Connection pooling and statement pooling are available to an application only if these features are implemented by the infrastructure on which the application is running.The important point is that pooling mechanisms, either connection pooling or statement pooling, operate purely in the background and do not in any way affect application code.



6.3 JDBC 3.0 New features

The JDBC 3.0 specification provides for statement pooling by defining a set of standard properties that apply to PooledConnection objects. These properties specify how many physical connections (PooledConnection objects) should be in the initial pool, how many should always be available, how long they can be idle before being closed, and so on.

The property that applies to statement pooling,maxStatements, specifies the total number of PreparedStatement objects that a PooledConnection object should keep open in its pool of statements. The logical diagram of pooled connection as follows,

6.4 Closing a Pooled Statement

An application must close a pooled statement in order for the statement to be available for reuse. An application can close a Statement object (and therefore a PreparedStatement object) in two ways

- by calling the close method on the statement

- calling the close method on the connection.

When an application calls the method PreparedStatement.close on a pooled statement, the logical statement it is using is closed, but the physical statement is made available for reuse in the statement pool.

Closing a connection closes not only the connection but also all of the statements created by that connection. Thus, when an application calls the method Connection.close on a pooled connection, it closes the Connection object it is using and also the logical PreparedStatement object it is using. However, the underlying physical connection, a PooledConnection object, is returned to the connection pool. Similarly, the physical PreparedStatement object is returned to the statement pool.

6.5 References

http://www.javaranch.com/journal/200601/JDBCConnectionPooling.html

http://www.java-samples.com/showtutorial.php?tutorialid=203

http://www.javaworld.com/javaworld/jw-10-2000/jw-1027-pool.html?page=2

http://download.oracle.com/docs/cd/E13222_01/wls/docs81/jconnector/connect.html