

Re: pooled connection myth

Source: <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2005-03/1838.html>

From: David McDivitt (*x12code-del_at_del-yahoo.com*)

Date: 03/17/05

Date: Thu, 17 Mar 2005 09:41:04 -0600

There is much to be gained by reusing known open connection objects, rather than issuing single use connection objects for each request. If connection objects are not modified by using transaction blocks, there's no reason not to share them between threads. The word "pooling" implies this, but pooling itself doesn't do very much. For the most part, all that happens is caching the database user name and password.

>From: Lee Fesperman <firstsql@ix.netcom.com>

>Subject: Re: pooled connection myth

>Date: Thu, 17 Mar 2005 00:35:07 GMT

>

>David McDivitt wrote:

>>

>> I want to send an email to people in my unit containing something similar to

>> the following and I would appreciate comments first:

>

>I'm not sure where you're getting your information, but I'd suggest you revise the
>email....

>

>> When calling the method `getPooledConnection`, it would imply you are getting
>> a truly pooled or shared connection. What you are getting is what the
>> framework wants you to have, and that is within the scope of present
>> implementation. Java is very robust and it's possible to connect to anything
>> and everything in the universe from one application. Certainly some of these
>> resource connections will be grouped together. In my investigation of
>> existent applications we have, grouping is only done at the resource driver
>> level. Individual database connections are not pooled or shared.

>>

>> To support my assertion, if `datasource.xml` is viewed in the struts
>> framework, the driver `COM.ibm.db2.jdbc.app.DB2Driver` can be found. This
>> driver only works one way. There is no secret, hidden, marvelous, behind the
>> scenes functionality present in the app server, or the DB2 database, when
>> this driver is used. There is no reusability of connection objects created
>> with this driver. If the close method is done on a connection object it
>> cannot be used again. It is history, toast, gone. Only a new connection can
>> be instantiated from the DB2 driver, which involves use of a user name and
>> password, again. If in code you find the close method being called on
>> connection objects, necessarily, those connection objects go away. Also, on

comp.lang.java.programmer: Re: pooled connection myth

>> a connection object, if transaction blocks are used, such apply to the whole
>> connection object. If threads do somehow share a connection and one starts a
>> transaction block, it would apply to the other thread as well. There is no
>> connection within a connection. The same goes for the autocommit property
>> (dirty writes).
>
>The general concept you're pushing is incorrect. First, connection pooling does work
>(I'll explain in a moment). Sharing a connection concurrently doesn't work, for the
>reasons you mentioned. Sharing it serially does work and is the basis of connection
>pooling.
>
>You're right that a regular JDBC driver doesn't support connection pooling at all.
>However, connection pooling software, like in an app server, solves this by wrapping the
>objects (Connection, Statement, ResultSet, ...) of the driver. This is easy because the
>important pieces of the driver are all implementations of interfaces.
>
>For instance, the connection pooling software uses its Connection wrapper to control
>calls to close(), so that close() doesn't close the connection. Instead, the wrapper
>close() returns the connection to the pool ... also doing necessary cleanup. There some
>other details, but that is the crux of it. The connection pooling software can pool
>connections but only allows one application to use it at a time. When the connection
>pooling software is through with the connection, it calls close() in the actual
>connection. A well-written set of wrappers prevents premature closing.
>
>The driver provider can provide more support for connection pooling by implementing
>javax.sql.ConnectionPoolDataSource and javax.sql.PooledConnection. In this case, the
>driver provider does its own wrapping in so many words and gives control to the app
>server through a listener mechanism, so the app server can do the actual pooling. JCA
>provides an enhanced form of this.
>
>This allows for a variety of implementations of the actual pooling mechanism.
>
>> Due to extensibility considerations, and potential grandiose use of
>> applications, the pooled connection concept might be appropriate, but is
>> dependent on certain java coding standards within an application and what is
>> done with connections. It cannot be assumed connections are pooled at a low
>> level simply because a method name says so.
>
>Yes, it can as I explained.
>
>> It is possible to code a singleton, run all connection requests through such
>> a connection manager, and simply return the same connection object each time
>> it's asked for. The struts framework does not do this for you. The
>> application must be coded that way, validity of the connection must be
>> tested at a timed interval in a separate thread, and no methods can be
>> called anywhere to modify the connection. A good connection manager might
>> use three connection objects and return them round robin to callers.
>
>This is not necessary.
>
>> Not understanding connection functionality and assuming certain

comp.lang.java.programmer: Re: pooled connection myth

>> *functionality exists on faith contributes to database problems.*

>

> *Sorry, you're off the track here.*

>

> *If my explanation was not clear enough, feel free to ask for clarification.*