

# Re: ConnectionPoolDataSource + MM MySQL + Tomcat4

**Source:** <http://coding.derkeiler.com/Archive/Java/comp.lang.java.programmer/2004-12/2604.html>

---

*stratfan\_at\_mindspring.com*

**Date:** 12/24/04

Date: 23 Dec 2004 18:42:48 -0800

I have encountered very similar problems / questions trying to get this to work with Tomcat 5.0.28 and MySQL or Oracle. I have probably spent two weeks getting this to work and JUST NOW got it to work while preparing followup questions to your question. Searching for web/newsgroup help on this topic seems to turn up contradictory examples.

## Approach #1

=====

- 1) define the core connection attributes in a <Resource> and <ResourceParams> object in the server.xml config of Tomcat and use javax.sql.DataSource as the <Resource> type and javax.sql.DataSource as the <factory> name in the <ResourceParams>
- 2) define a <resource-ref> object in your servlet web.xml config file that points to the info in the server.xml file
- 3) in your servlet's startup code, retrieve the <resource-ref> from the web.xml file as a DataSource object and preserve it as a static variable for the servlet
- 4) in functions needing DB access, use the DataSource variable to create a Connection
- 5) use the connection to do your SQL voodoo and process results
- 6) close your result sets, callable statements and connections to return the connection to the pool managed by Tomcat and avoid memory leaks

## Approach #2

=====

- 1) define the core connection attributes in a <Resource> and <ResourceParams> object in the server.xml config of Tomcat and use javax.sql.ConnectionPoolDataSource as the <Resource> type and javax.sql.ConnectionPoolDataSource (???) as the <factory> name in the <ResourceParams>
- 2) define a <resource-ref> object in your servlet web.xml

config file that points to the info in the server.xml file

3) in your servlet's startup code, retrieve the <resource-ref> from the web.xml file as a ConnectionPoolDataSource object then use .getPooledConnection to create a PooledConnection variable and preserve that PooledConnection as a static variable for the servlet

4) in functions needing DB access, use the PooledConnection variable to create a Connection via .getConnection()

5) use the connection to do your SQL voodoo and process results

6) close your result sets, callable statements and connections to return the connection to the pool managed by Tomcat and avoid memory leaks

### Approach #3

=====

(same as #2 but use vendor specific \_\_\_\_ConnectionPoolDataSource and \_\_\_\_ConnectionPool classes instead of the generic Database Connection Pool (DBCP) classes provided with Tomcat / Java.

I have found that if you

1) stick with using the generic DataSource class when defining your connection attributes in server.xml and web.xml

2) stick with using the generic DataSource class in your startup function to retrieve the attributes for use within the app

3) do not try to specify a <factory> for the DataSource in server.xml

4) make sure you have commons-dbcp-1.2.1.jar and commons-pool-1.2.jar in your \$CATALINA\_HOME/common/lib directory so Tomcat can see them and in your CLASSPATH so your classes will compile (NOTE: your version of Tomcat may have older versions of these files but the names should be similar)

things seem to work (at least with MySQL, still waiting to try this with Oracle at work).

My code snippets are provided below if they help. I HIGHLY recommend you include all of the advanced settings in your <ResourceParams> configuration even if you don't think you need them right away. You'll probably need them eventually and you'll have them all commented and available without more hunting later.

(signed)

stratfan@mindspring.com

PS --- these config / code samples look much better in long line format rather than the compressed format on the newsgroup viewer...

CONTENTS IN web.xml

=====

<resource-ref>

```
<description>Reference to Test MySQL Database for Connection Pool
based access</description>
<res-ref-name>jdbc/mysqlpool2</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

#### CONTENTS in server.xml

```
=====
<Resource name="jdbc/mysqlpool2" auth="Container"
type="javax.sql.DataSource"/>
<ResourceParams name="jdbc/mysqlpool2">
<parameter> <name>username</name>
<value>myusername</value> </parameter>
<parameter> <name>password</name>
<value>mypassword</value> </parameter>
<parameter> <name>driverClassName</name>
<value>org.gjt.mm.mysql.Driver</value> </parameter>
<parameter> <name>url</name>
<value>jdbc:mysql://localhost:3306/genapp</value> </parameter>
<!-- maxActive = max # of active connections to allow in
pool (0=no limit) -->
<!-- maxIdle = max # of idle connections to allow in
pool (0=no limit) -->
<!-- maxWait = max time in ms to wait for DB reply
before returning error -->
<!-- removeAbandoned = true means track leaked /
abandoned connections and kill them -->
<!-- removeAbandonedTimeout = time to wait before killing
old connections -->
<parameter> <name>maxActive</name> <value>10</value>
</parameter>
<parameter> <name>maxIdle</name> <value>30</value>
</parameter>
<parameter> <name>maxWait</name>
<value>10000</value> </parameter>
<parameter> <name>removeAbandoned</name>
<value>true</value> </parameter>
<parameter> <name>removeAbandonedTimeout</name>
<value>300</value> </parameter>
<parameter> <name>logAbandoned</name>
<value>true</value> </parameter>
</ResourceParams>
```

#### STARTUP CODE IN MY SERVLET CLASS

```
=====
try {
javax.naming.Context initialCtx = new InitialContext();
DataSource mysqlpoolDS = (DataSource)
initialCtx.lookup("java:comp/env/jdbc/mysqlpool2");
System.out.println("myapp loadConfiguration -- retrieved DataSource
```

```
jdbc/mysqlpool2");
Connection localConnection = mysqlpoolDS.getConnection();
System.out.println("myapp loadConfiguration -- opened local
connection from pool to jdbc/mysqlpool");

try {
PreparedStatement myselect =
localConnection.prepareStatement("select * from userlogins");
ResultSet results = null;
results = myselect.executeQuery();
if (results.first()) {
System.out.println("OK: "+ results.getInt ("login_id") + " "
+ results.getString("login") + " "
+ results.getString("firstname") );
}
}
catch (Exception theE) {
System.out.println("saw Exception doing SELECT:
"+theE.getMessage());
}
localConnection.close();
}
catch (Exception theE) {
System.out.println("myapp loadConfiguration -- Exception for
DataSource jdbc/mysqlpool2: " + theE);
}
```

```
catalina.out ENTRIES CONFIRMING POOLING IS WORKING
=====
myapp loadConfiguration -- retrieved DataSource jdbc/mysqlpool2
AbandonedObjectPool is used
(org.apache.commons.dbcp.AbandonedObjectPool@164b09c)
LogAbandoned: true
RemoveAbandoned: true
RemoveAbandonedTimeout: 300
extauth loadConfiguration -- opened local connection from pool to
jdbc/mysqlpool
OK: 1 joeblow Joe
```