Tomcat12

How to use BTM as the transaction manager in Tomcat 5.5.x

These instructions have been verified against BTM 1.1.

> **This is for BTM 1.2 and earlier**
>
> These instructions are for BTM 1.2 and earlier only. BTM 1.3 instructions are here.

Contents

- Step 1: Copy the BTM jars
- Step 2: Configure BTM as the transaction manager
- Step 3: Configure datasources that are transaction aware
- Step 4: Configure transaction manager and datasources initialization in your META-INF/context.xml
- Step 5: Configure datasources references in your web.xml

Step 1: Copy the BTM jars

Copy the following jars from the BTM distribution to the Tomcat `common/lib/` directory:

- btm-1.1.jar
- geronimo-spec-jta-1.0.1B-rc4.jar
- slf4j-jdk14.jar (*or any other one available here*)

Step 2: Configure BTM as the transaction manager

Windows: Create a file named `setenv.bat` with the following commands under Tomcat's `bin/` directory:

```
set CATALINA_OPTS=-Dbtm.root=%CATALINA_HOME%
-Dbitronix.tm.configuration=%CATALINA_HOME%\
conf\btm-config.properties
```

Unix: Create a file named `setenv.sh` with the following commands under Tomcat's `bin/` directory:

```
CATALINA_OPTS="-Dbtm.root=$CATALINA_HOME
-Dbitronix.tm.configuration=$CATALINA_HOME/c
onf/btm-config.properties"
```

Now create a file named `btm-config.properties` with the following properties under Tomcat's `conf/` directory:

```
bitronix.tm.serverId=tomcat-btm-node0
bitronix.tm.journal.disk.logPart1Filename=${
btm.root}/work/btm1.tlog
bitronix.tm.journal.disk.logPart2Filename=${
btm.root}/work/btm2.tlog
bitronix.tm.resource.configuration=${btm.roo
t}/conf/resources.properties
```

Finally, create an empty file named `resources.properties` under Tomcat's `conf/` directory.

Step 3: Configure datasources that are transaction aware

You have to put your datasources configurations in Tomcat's `conf/resources.properties` file. Here's an example of using BTM with a DataSource that implements javax.sql.XADataSource:

```
resource.ds1.className=org.apache.derby.jdbc
.EmbeddedXADataSource
resource.ds1.uniqueName=jdbc/mydatasource
resource.ds1.minPoolSize=0
resource.ds1.maxPoolSize=5
resource.ds1.driverProperties.databaseName=d
b1
resource.ds1.driverProperties.createDatabase
=create
```

This will create a `bitronix.tm.resource.jdbc.PoolingDataSource` that implements `javax.sql.DataSou rce` and interacts with the `javax.sql.XADataSource` provided in this instance by Derby.

If your database vendor does not provide an `XADataSource`, you can use BTM's `bitronix.tm.resource.jdbc .lrc.LrcXADataSource` as the `XADataSource` to allow your database connections to be controlled by the transaction manager:

```
resource.ds2.className=bitronix.tm.resource.
jdbc.lrc.LrcXADataSource
resource.ds2.uniqueName=jdbc/exampleNonXADS
resource.ds2.minPoolSize=0
resource.ds2.maxPoolSize=5
resource.ds2.driverProperties.driverClassNam
e=org.apache.derby.jdbc.EmbeddedDriver
resource.ds2.driverProperties.url=jdbc:derby
::db2;create=true
```

Again, we've used Derby as an example, but as the `LrcXADataSource` uses only the class name and url of a `jav a.sql.Driver`, you can use it with any database providing a JDBC driver.

Step 4: Configure transaction manager and datasources initialization in your META-INF/context.xml

In the web application where you want one or more datasource to be used, you have to create a META-INF/context.xml file.

```
<Context>

    <Resource name="jdbc/mydatasource"
auth="Container" type="javax.sql.DataSource"

factory="bitronix.tm.resource.ResourceObject
Factory" uniqueName="jdbc/mydatasource" />

    <Resource name="jdbc/exampleNonXADS"
auth="Container" type="javax.sql.DataSource"

factory="bitronix.tm.resource.ResourceObject
Factory" uniqueName="jdbc/exampleNonXADS" />

    <Transaction
factory="bitronix.tm.BitronixUserTransaction
ObjectFactory" />

</Context>
```

The `<Resource>` tags will bind a bitronix.tm.resource.ResourceObjectFactory object each, passing it a javax.naming.Reference containing a javax.naming.StringRefAddr containing the datasource's `uniqueName` as `addrType`.

> **Tomcat specific**
>
> This mechanism is internal to Tomcat. You do not have to worry about how it works, the `bitronix.tm.resource.ResourceObjectFactory` class will handle those details.

The `bitronix.tm.resource.ResourceObjectFactory` class will return the datasource previously configured in in Tomcat's `conf/resources.properties` with the specified `uniqueName` when it is fetched from JNDI.

The `<Transaction>` tag will just bind the transaction manager at the standard JNDI location `java:comp/UserTransaction`.

Step 5: Configure datasources references in your web.xml

Before your code can access configured datasources via JNDI ENC URLs, you need to declare resource references in your `web.xml`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <resource-env-ref>

<resource-env-ref-name>jdbc/mydatasource</re
source-env-ref-name>

<resource-env-ref-type>javax.sql.DataSource<
/resource-env-ref-type>
  </resource-env-ref>

  <resource-env-ref>

<resource-env-ref-name>jdbc/exampleNonXADS</
resource-env-ref-name>

<resource-env-ref-type>javax.sql.DataSource<
/resource-env-ref-type>
  </resource-env-ref>
</web-app>
```

Now you can perform JNDI lookups on those URLs to access the configured datasources:

```
DataSource exampleNonXADS =
ctx.lookup("java:comp/env/jdbc/exampleNonXAD
S");
DataSource mydatasource =
ctx.lookup("java:comp/env/jdbc/mydatasource"
);
```