JOTM

How To Use JOTM as the XA Transaction Manager in Jetty

These instructions have been tested with JOTM 2.0.10.

Step 1: Copy the jars

Assuming you have successfully downloaded JOTM, copy the following jars to jetty's `lib`/ext directory:

- connector-1_5.jar
- howl.jar
- jotm.jar
- jotm_jrmp_stubs.jar
- jta-spec1_0_1.jar
- jts1_0.jar
- ow_carol.jar
- xapool.jar

> ✅ **Logging**
>
> Depending on how you want your logging configured, you may need to also copy `jcl104-over-slf4j.jar` and an slf4j log implementation such as `slf4j-simple.jar` into the `lib/` directory. You should find both of these files in `lib/jsp-2.0`. You would need to do this if you are using jdk-1.5 and you want to use slf4j logging. Alternatively, if you are using jdk1.4 and/or you want to use commons-logging, you will need to copy the commons-logging jar and a commons logging impl into the `lib/` directory.

Step 2: Configure CAROL

In your jetty installation, create the file `resources/carol.properties` and edit its contents to contain these lines:

```
carol.start.ns=false
carol.start.jndi=false
carol.protocols=jrmp
carol.start.rmi=false
carol.jvm.rmi.local.call=true
carol.jndi.java.naming.factory.url.pkgs=org.mortbay.naming
```

Without this step, CAROL will assume control of JNDI from jetty and `java:comp/env` will not be set up correctly.

Step 3: Configure the transaction manager and datasources

You need to register an XA transaction manager and XA aware DataSources. There is more information about

jetty's JNDI facilities that you may find useful. Here are the snippets for your jetty config file, typically `etc/jetty.xml`. In this example, we will configure a Derby JDBC driver, but of course you can substitute your own.

```
<!-----------------------------------------
-----------------------------------------
-->
<!-- Configure a Jotm instance which
provides a
javax.transaction.TransactionManager     -->
<!-- and a javax.transaction.UserTransaction
implementation.
-->
<!-----------------------------------------
-----------------------------------------
-->
<New id="jotm"
class="org.objectweb.jotm.Jotm">
  <Arg type="boolean">True</Arg>
  <Arg type="boolean">False</Arg>
  <Call id="tm"
name="getTransactionManager"/>
  <Call id="ut" name="getUserTransaction"/>
</New>

<!-----------------------------------------
-----------------------------------------
-->
<!-- Set up the UserTransaction impl from
JOTM as the transaction manager for jetty
-->
<!-----------------------------------------
```

```
-----------------------------------------------
-->
  <New
class="org.mortbay.jetty.plus.naming.Resourc
e">
    <Arg></Arg>

<Arg>javax.transaction.TransactionManager</A
rg>
    <Arg><Ref id="ut"/></Arg>
  </New>

  <New id="tx"
class="org.mortbay.jetty.plus.naming.Transac
tion">
   <Arg>
```

```
      <Ref id="ut"/>
    </Arg>
  </New>
```

At this point, you have JOTM acting as the transaction manager. Now, you can define XA-aware DataSources. If you want them defined globally for all webapps within your jetty installation, you can put them inside `etc/jetty.xml`. Here's one example:

```
<!-----------------------------------------
-----------------------------------------
-->
<!-- Set up a DataSource that is XA aware.
JOTM uses XAPool for this.
-->
<!-----------------------------------------
-----------------------------------------
-->
  <New
class="org.mortbay.jetty.plus.naming.Resourc
e">
    <Arg>myxadatasource</Arg>
    <Arg>
      <New id="myxadatasourceA"
class="org.enhydra.jdbc.standard.StandardXAD
ataSource">
        <Set
name="DriverName">org.apache.derby.jdbc.Embe
ddedDriver</Set>
        <Set
name="Url">jdbc:derby:myderbyDB1A;create=tru
e</Set>
```

```
        <Set name="User"></Set>
        <Set name="Password"></Set>
        <Set name="transactionManager"><Ref
id="tm"/></Set>
     </New>
    </Arg>
  </New>


  <New id="mydatasource"
class="org.mortbay.jetty.plus.naming.Resourc
e">
    <Arg>jdbc/mydatasource</Arg>
    <Arg>
     <New
class="org.enhydra.jdbc.pool.StandardXAPoolD
ataSource">
       <Arg><Ref
id="myxadatasourceA"/></Arg>
       <Set
name="DataSourceName">myxadatasource</Set>
     </New>
    </Arg>
   </New>

<!-- If you want to be able to set up more
references in webapp specific files -->
<!-- such as context deployment files and
WEB-INF/jetty-env.xml files, you     -->
<!-- need to save a reference to the JOTM tm
object:                              -->
```

```
<Call name="setAttribute">
```

```
    <Arg>tm</Arg>
    <Arg><Ref id="tm"/></Arg>
  </Call>
```

Now you can hookup a `web.xml` `resource-ref` entry for `jdbc/mydatasource` and then you'll be able to do lookups in your webapp of `java:comp/env/jdbc/mydatasource`.

```
<!-----------------------------------------
------------------------->
<!-- web.xml snippet
->
<!-----------------------------------------
------------------------->
  <resource-ref>

<res-ref-name>jdbc/mydatasource</res-ref-nam
e>

<res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
```

If instead you want to make DataSources that are restricted in scope to only a particular webapp, then you need to put the declaration in a context deployment file (ie a .xml file in $JETTY-HOME/contexts) *or* in a `WEB-INF/jetty-env.xml` file, and you need to ensure that the org.mortbay.jetty.plus.naming.Resource declaration for the StandardXAPoolDataSource is scoped to the webapp by including as the first argument a reference to the webapp. Here's an example from a xml context deployment file:

```
<Configure id='wac'
class="org.mortbay.jetty.webapp.WebAppContex
t">

<!-----------------------------------------
```

```
-------------------------->
<!-- Get the JOTM reference from the Server
setup in etc/jetty.xml -->
<!-----------------------------------------
------------------------->
<Property name="Server" id="Server">
    <Call id="tm" name="getAttribute">
      <Arg>tm</Arg>
    </Call>
</Property>

<!-----------------------------------------
------------------------->
<!-- Set up the DataSource and XA-aware
DataSources                -->
<!-----------------------------------------
------------------------->
 <New
class="org.mortbay.jetty.plus.naming.Resourc
e">
    <Arg>myxadatasource</Arg>
    <Arg>
      <New id="myxadatasourceA"
class="org.enhydra.jdbc.standard.StandardXAD
ataSource">
        <Set
name="DriverName">org.apache.derby.jdbc.Embe
ddedDriver</Set>
        <Set
name="Url">jdbc:derby:myderbyDB1A;create=tru
e</Set>
```

```
        <Set name="User"></Set>
        <Set name="Password"></Set>
        <Set name="transactionManager"><Ref
id="tm"/></Set>
     </New>
    </Arg>
  </New>


  <New id="mydatasource"
class="org.mortbay.jetty.plus.naming.Resourc
e">
    <Arg><Ref id='wac'/></Arg>
    <Arg>jdbc/mydatasource</Arg>
    <Arg>
     <New
class="org.enhydra.jdbc.pool.StandardXAPoolD
ataSource">
       <Arg><Ref
id="myxadatasourceA"/></Arg>
       <Set
name="DataSourceName">myxadatasource</Set>
     </New>
    </Arg>
   </New>

<!-----------------------------------------
------------------------->
<!-- If you want to also be able to define
more JOTM DataSources   -->
<!-- from a WEB-INF/jetty-env.xml file, then
```

```
you need to save the  -->
<!-- reference to the tm object into the
webapp:                    -->
<!-----------------------------------------
------------------------->
<Call name="setAttribute">
```

```
  <Arg>tm</Arg>
  <Arg><Ref id="tm"/></Arg>
</Call>
```

NOTE in the example above, that the first org.mortbay.jetty.plus.naming.Resource is actually declared to be in Server scope, by the absence of a webapp reference as the first argument. It is the *second* org.mortbay.jetty.plus.naming.Resource (ie the one that you will lookup in your code) that uses the reference to the webapp. Ideally, both would be webapp-specific, but JOTMs internal implementation of StandardXAPoolDataSource requires being able to lookup the StandardXADataSource by absolute name, and thus it is easiest to declare that in the Server scope.

For more information on scoping of JNDI entries, see the JNDI page. Also see the Atomikos page for information on an alternative XA manager that is a lot easier to configure.

✅ **Using XAPool**

You MUST wrap the `StandardXADataSource` in a `StandardXAPoolDataSource` because `StandardXADataSource` does not use the XAConnection if you call `getConnection()`, thus connections won't be involved in the XA transaction.