IBatis12

Integrating BTM with iBatis

iBatis can be integrated straight with any JTA transaction manager. These instructions have been verified against BTM 1.2.

Contents

- JTA datasources
  - Setting up a JNDI server
  - API way: Creating the datasources
  - Resource Loader way: Creating the datasources
- iBatis SqlMapClients
  - Datasource JNDI location
  - Transaction Manager JNDI location
  - SqlMapConfig XML configuration files
- End result
  - Application code
- Download

JTA datasources

iBatis cannot directly create a BTM `PoolingDataSource`. You will have to create them yourself (either via the API or the Resource Loader) and make sure they are bound to a JNDI server.

Setting up a JNDI server

You have to bind the datasources to some JNDI server. You can use any one you wish, including the one embedded in your application server.

In the case you want to run iBatis in a standalone application, you can use Tomcat's JNDI server as it is very easy to use it in a standalone J2SE application. Just create a `jndi.properties` file at the root of your classpath. It should only contain this line:

```
java.naming.factory.initial=org.apache.naming.java.javaURLContextFactory
```

Then add these two jar files to your classpath:

```
naming-factory.jar
naming-resources.jar
```

They are available in any Tomcat 5.5 distribution, just look into the `TOMCAT_HOME/common/lib` folder.

This will configure Tomcat's JNDI server as the default implementation. It is a simple intra-VM server that implements the most important JNDI features. You can now just create a InitialContext with the no-args constructor to have access to it.

API way: Creating the datasources

As you can expect, you will need to create one `PoolingDataSource` per database. Say that you want to use two Embedded Derby databases, and configure them via the BTM API. Here is what your code would look like:

```
PoolingDataSource ds1 = new
PoolingDataSource();
ds1.setUniqueName("jdbc/testDS1");
ds1.setClassName("org.apache.derby.jdbc.Embe
ddedXADataSource");
ds1.setMaxPoolSize(3);
ds1.getDriverProperties().put("databaseName"
, "users1");
ds1.init();

PoolingDataSource ds2 = new
PoolingDataSource();
ds2.setUniqueName("jdbc/testDS2");
ds2.setClassName("org.apache.derby.jdbc.Embe
ddedXADataSource");
ds2.setMaxPoolSize(3);
ds2.getDriverProperties().put("databaseName"
, "users2");
ds2.init();
```

✅ **Datasource's unique name and JNDI location correspondence**

It is usually a good idea to set a datasource's unique name to the same value as its JNDI location although this is not mandatory.

Finally, here is the code to bind the datasources:

```
Context ctx = new InitialContext();
ctx.createSubcontext("jdbc");
ctx.rebind("jdbc/testDS1", ds1);
ctx.rebind("jdbc/testDS2", ds2);
ctx.close();
```

Resource Loader way: Creating the datasources

You can use BTM's Resource Loader instead of the BTM API. It is usually a good idea when you want to create a fully standalone application as you can get rid of the datasources creation, JNDI binding and shutdown code.

Create a `datasources.properties` file in the current directory containing these properties:

```
bitronix.tm.resource.bind=true

resource.ds1.className=org.apache.derby.jdbc
.EmbeddedXADataSource
resource.ds1.uniqueName=jdbc/testDS1
resource.ds1.maxPoolSize=3
resource.ds1.driverProperties.databaseName=u
sers1

resource.ds2.className=org.apache.derby.jdbc
.EmbeddedXADataSource
resource.ds2.uniqueName=jdbc/testDS2
resource.ds2.maxPoolSize=3
resource.ds2.driverProperties.databaseName=u
sers2
```

Resource Loader JNDI binding

When you configure the Resource Loader to bind datasources to JNDI (by setting the `bitronix.tm.resource.bind` property to `true`) it will bind them using the unique name as the JNDI location.

In your application code, you will have to configure BTM to use the resource loader:

```
TransactionManagerServices.getConfiguration().setResourceConfigurationFilename("./datasources.properties");
userTransaction = TransactionManagerServices.getTransactionManager();
```

This has the exact same behavior as creating the `PoolingDataSource` objects and binding them to JNDI yourself. It is just more convenient.

iBatis SqlMapClients

You need to configure exactly one `SqlMapClient` per datasource.

Datasource JNDI location

You have to tell iBatis where to get the BTM datasource via JNDI. Add a `DataSource` property to the `dataSource` tag and set its value to the JNDI location of your datasource:

```
<dataSource type="JNDI">
  <property name="DataSource" value="jdbc/testDS1"/>
</dataSource>
```

Transaction Manager JNDI location

You have to set `transactionManager` tag's type attribute to `JTA` and wrap the `dataSource` tag in it.

```
<transactionManager type="JTA" >
    <property name="UserTransaction"
value="UserTransaction"/>
    <dataSource type="JNDI">
      <property name="DataSource"
value="jdbc/testDS1"/>
    </dataSource>
  </transactionManager>
```

SqlMapConfig XML configuration files

Here is what the `SqlMapConfig1.xml` file will look like for the first datasource.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL
Map Config 2.0//EN"

"http://ibatis.apache.org/dtd/sql-map-config
-2.dtd">

<sqlMapConfig>

  <transactionManager type="JTA" >
    <property name="UserTransaction"
value="UserTransaction"/>
    <dataSource type="JNDI">
      <property name="DataSource"
value="jdbc/testDS1"/>
    </dataSource>
  </transactionManager>

  <sqlMap
resource="bitronix/examples/ibatis/entities/
User.xml"/>

</sqlMapConfig>
```

And here is the `SqlMapConfig2.xml` for the second datasource:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL
Map Config 2.0//EN"

"http://ibatis.apache.org/dtd/sql-map-config
-2.dtd">

<sqlMapConfig>

  <transactionManager type="JTA" >
    <property name="UserTransaction"
value="UserTransaction"/>
    <dataSource type="JNDI">
      <property name="DataSource"
value="jdbc/testDS2"/>
    </dataSource>
  </transactionManager>

  <sqlMap
resource="bitronix/examples/ibatis/entities/
User.xml"/>

</sqlMapConfig>
```

End result

Now that iBatis and BTM are properly configured, you can simply use the JTA and iBatis APIs in your application.

Application code

Here is what your code will look like when you want to update the content of both databases atomically:

```
for (int i=0; i<10 ;i++) {
    System.out.println("Iteration #" +
(i+1));

userTransaction.setTransactionTimeout(60);
    userTransaction.begin();

    try {
        System.out.println("*** DB1 ***");
        persistUser(smc1, "user");
        listUsers(smc1);

        System.out.println("*** DB2 ***");
        persistUser(smc2, "user");
        listUsers(smc2);

        userTransaction.commit();
    }
    catch (Exception ex) {
        ex.printStackTrace();
        userTransaction.rollback();
    }
}
```

Say that `persistUser()` creates a new user, in *no way* will a user be created in one database and not in the other.

Download

You can download a sample runnable application putting these explanations in practice. It contains all the code that has been skipped for clarity in this page. Both the API and Resource Loader ways are implemented so you can try both and see which one you prefer.

You can download this demo here: iBatisBTM12.zip.

There is an ant **build.xml** file included as well as a the necessary batch and shell scripts required to run the application from Windows or Unix.

Before you run the application, you have to create the Derby database. Just run the included **derby-create.sh** or **derby-create.bat** script to do so, it will create two directories called **users1** and **users2**. Then you can start the demo by either running **run_api.sh** or **run_api.bat** for the API version, **run_rl.sh** or **run_rl.bat** for the Resource Loader version.

Here is the list of JAR files with version required to run this demo. They're all included in the downloadable ZIP file.

JAR name	Version
btm-1.2.jar	BTM 1.2
geronimo-spec-jta-1.0.1B-rc4.jar	BTM 1.2
slf4j-api-1.4.3.jar	SLF4J 1.4.3
slf4j-jdk14-1.4.3.jar	SLF4J 1.4.3
derby-10.3.1.4.jar	Derby 10.3.1.4
derbytools-10.3.1.4.jar	Derby 10.3.1.4
commons-logging-1.0.4.jar	Apache Commons Logging 1.0.4
ibatis-2.3.3.720.jar	iBatis 2.3.3
naming-factory.jar	Tomcat 5.5.23
naming-resources.jar	Tomcat 5.5.23