Hibernate

Integrating BTM with Hibernate

Hibernate can be integrated straight with any JTA transaction manager. These instructions have been verified against BTM 2.1.1 and Hibernate 3.6.3.Final.

The biggest added value (omitting the fact that you can use Hibernate and two databases) is Hibernate's Current Session context management with JTA. You do not have to take care about opening nor closing `Session` as Hibernate will automatically bind them to the JTA transaction's lifecycle. You just have to make sure JTA transactions are properly started and ended.

> **JPA, Hibernate and BTM**
>
> The example discussed here uses the Hibernate API but the JPA API could be used as well. You just need to use the EntityManager but the same configuration applies.

Contents

- JTA datasources
  - Setting up the BTM JNDI server
  - API way: Creating the datasources
  - Resource Loader way: Creating the datasources
- Hibernate Session factories
  - Datasource JNDI location
  - Current session context
  - Transaction factory class
  - Transaction manager lookup class
  - SessionFactory XML configuration files
- End result
  - Application code
- Download

JTA datasources

Hibernate cannot directly create a BTM `PoolingDataSource`. You will have to create them yourself (either via the API or the Resource Loader).

Setting up the BTM JNDI server

You have to bind the datasources and the transaction manager to some JNDI server. You can use any one you wish, but BTM 2.x.x ships with one you might find more convenient to use.

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=bitronix.tm.jndi
.BitronixInitialContextFactory
```

You can now just create a InitialContext with the no-args constructor to have access to it.

> **Multiple JNDI providers**
>
> If you are running your application in a context in which there already is a JNDI provider installed (for instance in a web application) you can tell Hibernate to perform lookups in a specific JNDI environment instead by setting the hibernate.jndi.class property to `bitronix.tm.jndi.BitronixInitialContextFactory` in Hibernate's configuration instead.

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

The BTM JNDI provider will automatically bind the datasources under their unique name. In this case, you can look up jdbc/testDS1 or jdbc/testDS2 as soon as the transaction manager started without having anything else to configure.

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 and shutdown code.

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

```
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 and JNDI binding

As with the API, the datasources will be available in JNDI under their unique name.

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

```
TransactionManagerServices.getConfiguration(
).setResourceConfigurationFilename("./dataso
urces.properties");
userTransaction =
TransactionManagerServices.getTransactionMan
ager();
```

This has the exact same behavior as creating the `PoolingDataSource` objects yourself. It is just more convenient.

Hibernate Session factories

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

Datasource JNDI location

You have to tell Hibernate where to get the BTM datasource via JNDI. Add a connection.datasource property and set its value to the JNDI location of your datasource:

```
<property name="connection.datasource">jdbc/testDS1</property>
```

Current session context

You have to set current_session_context_class to `jta`.

```
<property name="current_session_context_class">jta</property>
```

Transaction factory class

You have to set transaction.factory_class to org.hibernate.transaction.JTATransactionFactory.

> **JPA / EJB3 API**
>
> When you use Hibernate via the JPA / EJB3 API (also known as *Hibernate EntityManager*) you should not set the `transaction.factory_class` property as the `Ejb3Configuration` class will configure a special one which is JTA compatible if you configure your `PersistenceUnit`'s transaction type to JTA.

```
<property name="transaction.factory_class">org.hibernate.transaction.JTATransactionFactory</property>
```

Transaction manager lookup class

You have to set transaction.manager_lookup_class to an implementation of TransactionManagerLookup. Hibernate ships with one that can lookup BTM since version 3.3.

add this property to your config to use it:

```
<property
name="transaction.manager_lookup_class">org.
hibernate.transaction.BTMTransactionManagerL
ookup</property>
```

BTMTransactionManagerLookup and JNDI

The BTMTransactionManagerLookup implementation which ships with Hibernate has a limitation regarding JNDI lookup of the transaction manager. If you're going to use it within an application server that already contains a JNDI server you might encounter problems.
An improved version which solves that issue has been submitted to the Hibernate project, see: http://opensource.atlassian.com/projects/hibernate/browse/HHH-3739
Also remember to change the jndiUserTransactionName value to something else, for instance: ***bitronixTransactionManager***.

SessionFactory XML configuration files

Here is what the `hibernate_testDS1.cfg.xml` file will look like for the first datasource. Some other mandatory properties also have to be added, like the `dialect`, `cache.provider_class` and of course the required object mappings.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-conf
iguration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <property
name="connection.datasource">jdbc/testDS1</p
```

```
roperty>
        <property
name="connection.release_mode">after_stateme
nt</property>
        <property
name="current_session_context_class">jta</pr
operty>
        <property
name="transaction.factory_class">org.hiberna
te.transaction.JTATransactionFactory</proper
ty>
        <property
name="transaction.manager_lookup_class">org.
hibernate.transaction.BTMTransactionManagerL
ookup</property>

        <property
name="dialect">org.hibernate.dialect.DerbyDi
alect</property>
        <property
name="cache.provider_class">org.hibernate.ca
che.NoCacheProvider</property>
        <property
name="show_sql">true</property>

        <mapping
resource="bitronix/examples/hibernate/entiti
es/User.hbm.xml"/>
```

```
    </session-factory>

</hibernate-configuration>
```

And here is the `hibernate_testDS2.cfg.xml` for the second datasource:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-conf
iguration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <property
name="connection.datasource">jdbc/testDS2</p
roperty>
        <property
name="connection.release_mode">after_stateme
nt</property>
        <property
name="current_session_context_class">jta</pr
operty>
        <property
name="transaction.factory_class">org.hiberna
te.transaction.JTATransactionFactory</proper
ty>
        <property
```

```
name="transaction.manager_lookup_class">org.
hibernate.transaction.BTMTransactionManagerL
ookup</property>

        <property
name="dialect">org.hibernate.dialect.DerbyDi
alect</property>
        <property
name="cache.provider_class">org.hibernate.ca
che.NoCacheProvider</property>
        <property
name="show_sql">true</property>

        <mapping
resource="bitronix/examples/hibernate/entiti
es/User.hbm.xml"/>
```

```
    </session-factory>

</hibernate-configuration>
```

End result

Now that Hibernate and BTM are properly configured, you can simply use the JTA and Hibernate 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(sf1, "user");
        listUsers(sf1);

        System.out.println("*** DB2 ***");
        persistUser(sf2, "user");
        listUsers(sf2);

        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: HibernateBTM2x.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-2.1.1.jar	BTM 2.1.1
geronimo-jta_1.1_spec-1.1.1.jar	BTM 2.1.1
slf4j-api-1.6.1.jar	SLF4J 1.6.1
slf4j-jdk14-1.6.1.jar	SLF4J 1.6.1
derby-10.3.1.4.jar	Derby 10.3.1.4
derbytools-10.3.1.4.jar	Derby 10.3.1.4
antlr-2.7.6.jar	Hibernate 3.6.3.Final
hibernate-jpa-2.0-api-1.0.0.Final.jar	Hibernate 3.6.3.Final
javassist-3.12.0.GA	Hibernate 3.6.3.Final
commons-collections-3.1.jar	Hibernate 3.6.3.Final
dom4j-1.6.1.jar	Hibernate 3.6.3.Final
hibernate3.jar	Hibernate 3.6.3.Final