OJB - Connection Handling

by Armin Waibel

1. Introduction

In this section the connection handling within OJB will be described. OJB use two classes which share the connection management:

- `org.apache.ojb.broker.accesslayer.ConnectionFactory`
- `org.apache.ojb.broker.accesslayer.ConnectionManagerIF`

2. ConnectionFactory

The `org.apache.ojb.broker.accesslayer.ConnectionFactory` interface implementation is a pluggable component (via the OJB.properties file - more about the OJB.properties file here) responsible for creation/lookup and release of connections.

```
public interface ConnectionFactory
{
    Connection lookupConnection(JdbcConnectionDescriptor jcd) throws LookupException;

    void releaseConnection(JdbcConnectionDescriptor jcd, Connection con);

    void releaseAllResources();
}
```

To enable a specific *ConnectionFactory* implementation class in *OJB.properties* file, set property *ConnectionFactoryClass*:

```
ConnectionFactoryClass=org.apache.ojb.broker.accesslayer.ConnectionFactoryPooledImpl
```

OJB was shipped with a bunch of different implementation classes which can be used in different situations, e.g. creation of connection instances is costly, so pooling of connection will increase performance.

To make it more easier to implement own *ConnectionFactory* classes an abstract base class called `org.apache.ojb.broker.accesslayer.ConnectionFactoryAbstractImpl` exists, most shipped implementation classes inherited from this class.

Note:

All shipped implementation with support for connection pooling only pool direct obtained connections, *DataSources* will **never** be pooled.

2.1. ConnectionFactoryPooledImpl

An *ConnectionFactory* implementation using commons-pool to pool the requested connections. On lookup call a connection was borrowed from pool and returned on the release call. This implementation was used as *default* setting in OJB.properties file.

This implementation allows a wide range off different settings, more info about the configuration properties can be found in metadata repository *connection-pool* section.

2.2. ConnectionFactoryNotPooledImpl

The name is programm, this implementation creates a new connection on each request and close it on release call. All *connection-pool* settings are ignored by this implementation.

2.3. ConnectionFactoryManagedImpl

This is a specific implementation for use in *managed environments* like J2EE conform application server. In managed environments it is **mandatory** to use *DataSource* provided by the application server.

All *connection-pool* settings are ignored by this implementation.

2.4. ConnectionFactoryDBCPImpl

An implementation using commons-dbcp to pool the connections.

This implementation allows a wide range off different settings, more info about the configuration properties can be found in metadata repository *connection-pool* section.

3. ConnectionManager

The `org.apache.ojb.broker.accesslayer.ConnectionManagerIF` interface implementation is a pluggable component (via the OJB.properties file - more about the OJB.properties file here) responsible for managing the connection usage lifecycle and connection status (commit/rollback of connections).

```
public interface ConnectionManagerIF
{
    JdbcConnectionDescriptor getConnectionDescriptor();

    Platform getSupportedPlatform();

    boolean isAlive(Connection conn);

    Connection getConnection() throws LookupException;

    boolean isInLocalTransaction();

    void localBegin();

    void localCommit();

    void localRollback();

    void releaseConnection();

    void setBatchMode(boolean mode);

    boolean isBatchMode();

    void executeBatch();

    void executeBatchIfNecessary();

    void clearBatch();
}
```

The *ConnectionManager* was used by the *PersistenceBroker* to handle connection usage lifecycle.

4. Questions and Answers

4.1. How does OJB handle connection pooling?

OJB does connection pooling per default, expect for datasources. Datasources never will be pooled.

Responsible for managing the connections in OJB are implementations of the `org.apache.ojb.broker.accesslayer.ConnectionFactory.java` interface. There are several implementations shipped with OJB called `org.apache.ojb.broker.accesslayer.ConnectionFactoryXXXImpl.java`. You can find among other things a none pooling implementation and a implementation using jakarta-DBCP api.

To manage the connection pooling define in your *jdbc-connection-descriptor* a *connection-pool* element. Here you can specify the properties for the used *ConnectionFactory* implementation. More common info see repository section or in repository.dtd.

4.2. Can I directly obtain a java.sql.Connection within OJB?

The PB-api enabled the possibility to obtain a connection from the current used `PersistenceBroker` instance:

```
PersistenceBroker broker = PersistenceBrokerFactory.
createPersistenceBroker(myKey);
broker.beginTransaction();
// do something

Connection con = broker.serviceConnectionManager().getConnection();
// perform your connction action and do more

broker.commitTransaction();
broker.close();
```

Note:
Do not commit the connection instance, this will be done by OJB when PersistenceBroker commit-/abortTransaction was called.

If no transaction was running, it is possible to release connection after use by hand:

```
pBroker.serviceConnectionManager().releaseConnection();
```

This call cleanup the used connection and pass the instance to release method of *ConnectionFactory* (this will e.g. return connection it to pool or close it). If you don't do any connection cleanup at the latest the connection will be released on PB.close() call.

Note:
Never do a `Connection.close()` call on the obtained connection instance by hand!!
This will be handled by the *ConnectionFactory*.

Users who interested in this section also interested in 'Is it possible to perform my own sql-queries in OJB?'.