Chapter 1. Tracking JDBC Connection Leaks in Java EE Applications

When an application is managing itself the access to the database through JDBC Datasources, the release of JDBC resources needs to be done. In some cases, the close of the connection is not done. As the JOnAS application server is using a pool to manage these accesses to the database, it means that the pool may reach its maximum size as the connections are not closed (and then put back into the pool). And if the pool reach its maximum size, new requests will go in the wait state or wll be aborted. Thus, this kind of problems is a huge problem in a production system.

Fortunately, there are some features provided by JOnAS to handle this case.

- For example there are JDBC Pooling mechanisms that will kill connections if they've not be used since a long time (that can be configured) so the pool can lowered its size.
- There is also a new feature that will close automatically connections if they are not closed after their access. This feature is provided through the JDBC JNDI Interceptor which may be configured with `JONAS_BASE/conf/jndi-interceptors.xml` file. By default, all connections that are not closed will be closed automatically. This can be changed with the *forceClose* option. Also, by default this mechanism is applied on all datasources. This can be changed by updating the *regExp* element.
- JOnAS is providing informations that allows to track the root of the problem in the source code of the application. For example, when there is a connection leak, JOnAS is able to print in the log or to show in the JOnAS Admin console (With the JDBC Connection Leaks module) the line of code where the connection was opened. Thus, a quick review of the code needs to be done in order to know why the close() instruction has not be done.