Logging with log4jdbc

Log4JDBC is a JDBC proxy. Which means it can log almost all activity to do with the database, which is very helpful for troubleshooting.

You can log

- sql statements with bound parameters
- returned record sets (fairly expensive!)
- timing of sql statements

To visualize where log4jdc snaps in, here's a pseudo diagram:

(Retain) <====> Hibernate <===> Connection Pool (c3p0) <===> log4jdbc <===> native JDBC client driver <===> DB server.

As you can see, log4jdbc sits at a lower level than Hibernate or the connection pool, just above the JDBC client.

More Information

While I will discuss the basic configuration below, there's even more information at

https://code.google.com/p/log4jdbc/

Configuration

log4jdbc is totally inactive until activated as discussed below, and has no performance effect until activated. After activation it is likely to have moderate to significant performance effect depending on the system and the logging level.

There are two steps to setting up log4jdbc:

- Activation
- Logging configuration

Activation

Is fairly straightforward. Go to RetainServer/WEB-INF/cfg and open up ASConfig.cfg.

The two relevant lines are

<DBURL>jdbc:mysql://localhost/retain</DBURL>

<confDBURL>jdbc:mysql://localhost/retain</confDBURL>

The DBURL refers to the MESSAGE STORE, the confDBURL to the CONFIGURATION STORE. You may want to activate for one or both.

To activate simply insert the log4jdbc statement between the jdbc: and the dbidentifier. For example:

<DBURL>jdbc:log4jdbc:mysql://localhost/retain</DBURL>

On restart log4jdbc will be activated.

Logging Configuration

You wont see much until this is configured. By default all logging is commented out. When uncommented, all logging should go to a "diag" subdirectory of the logs file.

Open up log4j.properties in RetainServer/WEB-INF/classes

Skipping to the bottom you'll see a lot of items commented out (with # sign)

"There are 5 loggers that are used by log4jdbc...If any of the 5 logs are set to ERROR level or above (e.g ERROR, INFO or DEBUG) then log4jdbc will be activated, wrapping and logging activity in the JDBC connections returned by the underlying driver"

Let's break this statement down. There are FIVE different kinds of logging information that can be produced

jdbc.sqlonly Logs only SQL. SQL executed within a prepared statement is automatically shown with it's bind arguments replaced with the data bound at that position, for greatly increased readability.

jdbc.sqltiming Logs the SQL, post-execution, including timing statistics on how long the SQL took to execute.

jdbc.audit Logs ALL JDBC calls except for ResultSets. This is a very voluminous output, and is not normally needed unless tracking down a specific JDBC problem.

jdbc.resultset Even more voluminous, because all calls to ResultSet objects are logged.

jdbc.connection Logs connection open and close events as well as dumping all open connection numbers. This is very useful for hunting down connection leak problems.

You probably want to activate only one or two of these. As you can see from the descriptions some of them overlap.

So, having decided which LOGGER (s) you want (Step 1), we need to set the logging level (Step 2) and activate the appender (step 3).

This is best explained by example. Let's say we want jdbc.sqlonly and jdbc.connection to be on

First, we'd find these lines

```
! Log only the SQL that is executed.
#log4j.logger.jdbc.sqlonly=INFO,sql
#log4j.additivity.jdbc.sqlonly=false


! Log connection open/close events and connection number dump
#log4j.logger.jdbc.connection=FATAL,connection
#log4j.additivity.jdbc.connection=false
```

and we'd remove the #, ending up with

```
! Log only the SQL that is executed.
log4j.logger.jdbc.sqlonly=INFO,sql
log4j.additivity.jdbc.sqlonly=false


! Log connection open/close events and connection number dump
log4j.logger.jdbc.connection=FATAL,connection
log4j.additivity.jdbc.connection=false
```

We'd also change the logging level of the jdbc.connection to INFO or DEBUG

```
! Log only the SQL that is executed.
log4j.logger.jdbc.sqlonly=INFO,sql
log4j.additivity.jdbc.sqlonly=false


! Log connection open/close events and connection number dump
log4j.logger.jdbc.connection=INFO,connection
log4j.additivity.jdbc.connection=false
```

Next, we'd find the commented out "appenders". Appenders define where (file wise) and how the information is logged. They are commented out too, because otherwise we'd always have the diag folder created with 0 byte files, even with the loggers commented out.

To find them, look at these lines you just uncommented

```
log4j.logger.jdbc.sqlonly=INFO,sql
log4j.logger.jdbc.connection=INFO,connection
```

This says "log4j, please add a logger of the name jdbc.sqlonly. Use the logging level INFO, and the appender named sql.". Similarly the second line uses the appender named connection.

You'll find these further down:

```
! the appender used for the JDBC API layer call logging above, sql only
#log4j.appender.sql=org.apache.log4j.FileAppender
#log4j.appender.sql.File=${catalina.base}/logs/diag/sql.log
#log4j.appender.sql.Append=true
#log4j.appender.sql.layout=org.apache.log4j.PatternLayout
```

```
#log4j.appender.sql.layout.ConversionPattern=-----> %d{yyyy-MM-dd HH:mm:ss.SSS} %m%n%n
```
and
```
! the appender used for the JDBC Connection open and close events
#log4j.appender.connection=org.apache.log4j.FileAppender
#log4j.appender.connection.File=${catalina.base}/logs/diag/connection.log
#log4j.appender.connection.Append=true
#log4j.appender.connection.layout=org.apache.log4j.PatternLayout
#log4j.appender.connection.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss.SSS} %m%n
```

Do you see how this works? The appenders are always prepended with "log4j.appender" and then the appender name.

Uncomment these items and you are good to go.

Note the log4j.appender.<appenderName>.File -- that's where the log file will be named and stored.

To filter the log with more helpful output (like showing just the activity from your application), do the following:

LINUX:

1. Edit /etc/opt/beginfinite/retain/tomcat7/j2ee
2. Find the “JAVA_OPTS statement. Should be near line 96.

It reads: JAVA_OPTS="-Djava.library.path=$JAVA_BINDIR"

3. Add -Dlog4jdbc.debug.stack.prefix=com.gwava at the end, so that it reads:

JAVA_OPTS="-Djava.library.path=$JAVA_BINDIR
-Dlog4jdbc.debug.stack.prefix=com.gwava"

WINDOWS:

1. Configure Tomcat.
2. Java tab.
3. Under “Java Options”, add a line that reads:

-Dlog4jdbc.debug.stack.prefix=com.gwava

There are other options that can be added. See section labeled, “Options” at: https://code.google.com/p/log4jdbc/ . To add any of those options listed, preceed them with “-D”. For example, if adding the log4jdbc.debug.stack.prefix option, it should read -Dlog4jdbc.debug.stack.prefix.