log4jdbc

JDBC proxy driver for logging SQL and other interesting information.

log4jdbc is a Java JDBC driver that can log SQL and/or JDBC calls (and optionally SQL timing information) for other JDBC drivers using the Simple Logging Facade For Java (SLF4J) logging system.

This project was originally hosted on SourceForge.

Code license: Apache License 2.0

Labels: JDBC, SQL, logging, Java, driver, SLF4J, spy, proxy, performance, profiling, database, db

Featured downloads: Show all

log4jdbc-1.2alpha2.zip
log4jdbc3-1.2alpha2.jar
log4jdbc4-1.2alpha2.jar

Featured wiki pages: Show all

FAQ

Blogs: Arthur Blake's Blog.

Feeds: Project feeds

Groups: log4jdbc general support group

Project owners: People details

arthur.blake

News

2009-02-26:

log4jdbc 1.2 alpha 2 released: one bug fix and a couple of small new features...

2008-11-26: So long SourceForge... (and thanks for all the fish!) Moved to Google Code! Also, created a new google group for general support, feedback and questions.

2008-11-08: log4jdbc 1.2 alpha 1 released. Many new options for controlling SQL output and a new log for viewing connection open/close events (very useful for hunting down connection leak issues.) See CHANGES for all the release details.

2008-04-11: log4jdbc 1.1 final release out!

2007-11-10: log4jdbc 1.1 beta 1 released. New optional timing threshold settings for honing in on slow SQL.

2007-07-25: log4jdbc 1.1 alpha 2 released. JDBC 4 support!

2007-05-29: log4jdbc 1.1 alpha 1 released. Most notable change is that the Simple Logging Facade for Java is now used instead of log4j directly.

2007-04-21: log4jdbc 1.0 has been released! Download it and give it a try!

Features

- Full support for JDBC 3 and JDBC 4!
- Easy to configure, in most cases all you need to do is change the driver class name to **net.sf.log4jdbc.DriverSpy** and prepend **"jdbc:log4"** to your existing jdbc url, set up your logging categories and you're ready to go!
- In the logged output, for prepared statements, the bind arguments are automatically inserted into the SQL output. This greatly Improves readability and debugging for many cases.
- SQL timing information can be generated to help identify how long SQL statements take to run, helping to identify statements that are running too slowly and this data can be post processed with an included tool to produce profiling report data for quickly identifying slow SQL in your application..

- SQL connection number information is generated to help identify connection pooling or threading problems.
- Works with any underlying JDBC driver, with JDK 1.4 and above, and SLF4J 1.x.
- Open source software, licensed under the business friendly **Apache 2.0 license**:

  http://www.apache.org/licenses/LICENSE-2.0

Usage

Using data sources or maven? See the

FAQ.

1. Decide if you need JDBC 3 or JDBC 4 support.

- If you are using JDK 1.4 or 1.5, you should use the JDBC 3 version of log4jdbc.
- If you are using JDK 1.6 or 1.7, you should use the JDBC 4 version of log4jdbc (even if the actual underlying JDBC driver you are using is a JDBC 3 or older driver).

  Currently there are very few actual JDBC 4 drivers on the market. (The only major one that I really know about is Apache Derby aka The Java DB distributed with JDK 1.6.) JDBC 4 support was added with the JDK 1.6 release and adds many additional features over and above JDBC 3. However, the log4jdbc JDBC 4 driver can wrap a JDBC 3 or older driver and it's recommended that if you use JDK 1.6 or above, that you use the log4jdbc JDBC 4 driver that is compiled with JDK 1.6.

  Note that JDBC 2 is not currently supported by log4jdbc, although if you are using JDK 1.4 and above, the log4jdbc 3 or 4 driver should be able to wrap an older JDBC 2 driver as well-- log4jdbc just won't work with Java 1.3 and earlier.

  Choose and download one of the driver .jar files:

  - log4jdbc3-1.2alpha2.jar for JDBC 3 support in JDK 1.4 , JDK 1.5
  - log4jdbc4-1.2alpha2.jar for JDBC 4 support in JDK 1.6 , JDK 1.7

  Place the log4jdbc jar that you choose into your application's classpath.

2. Choose which java logging system you will use.

In many cases, you already know this, because it is dictated by your existing application. log4jdbc uses the Simple Logging Facade for Java (SLF4J) which is a very simple and very flexible little library that lets you pick among many common java logging systems:

- Log4j
- java.util logging in JDK 1.4
- logback
- Jakarta Commons Logging

SLF4J is designed to de-couple your application from the java logging system so you can choose any one you want. This is the same goal of Jakarta Commons Logging. However many people have had headaches and issues with classloading problems in complex environments using Jakarta Commons Logging. SLF4J solves these problems with it's much simpler design, and you can even integrate SLF4J to use Jakarta Commons Logging, if you really want to (or are required to) use it.

Download the latest official SLF4J release.

You will need two jars: slf4j-api-1.5.0.jar (or the latest available version) and whichever jar you pick depending on the java logging system you choose.

Place these two .jar files into your application's classpath.

Please read the documentation at the SLF4J website. It's really easy to set up!

3. Set your JDBC driver class to net.sf.log4jdbc.DriverSpy in your application's configuration.

The underlying driver that is being spied on in many cases will be loaded automatically without any additional configuration.

The log4jdbc "spy" driver will try and load the following popular jdbc drivers:

Driver Class	Database Type
oracle.jdbc.driver.OracleDriver	Oracle
com.sybase.jdbc2.jdbc.SybDriver	Sybase
net.sourceforge.jtds.jdbc.Driver	jTDS SQL Server & Sybase driver
com.microsoft.jdbc.sqlserver.SQLServerDriver	Microsoft SQL Server 2000 driver
com.microsoft.sqlserver.jdbc.SQLServerDriver	Microsoft SQL Server 2005 driver
weblogic.jdbc.sqlserver.SQLServerDriver	Weblogic SQL Server driver
com.informix.jdbc.IfxDriver	Informix
org.apache.derby.jdbc.ClientDriver	Apache Derby client/server driver, aka the Java DB
org.apache.derby.jdbc.EmbeddedDriver	Apache Derby embedded driver, aka the Java DB
com.mysql.jdbc.Driver	MySQL
org.postgresql.Driver	PostgresSQL
org.hsqldb.jdbcDriver	HSQLDB pure Java database
org.h2.Driver	H2 pure Java database

If you want to use a different underlying jdbc driver that is not already in the above supported list, set a system property, **log4jdbc.drivers** to the class name of the additional driver. This can also be a comma separated list of driver class names if you need more than one.

```
-Dlog4jdbc.drivers=<driverclass>[,<driverclass>...]
```

4. Prepend jdbc:log4 to the normal jdbc url that you are using.

For example, if your normal jdbc url is

```
jdbc:derby://localhost:1527//db-derby-10.2.2.0-bin/databases/MyDatabase
```

then You would change it to:

```
jdbc:log4jdbc:derby://localhost:1527//db-derby-10.2.2.0-bin/databases/MyDat
```

to use log4jdbc.

5. Set up your loggers.

There are 5 loggers that are used by log4jdbc, If all 5 are set to a level less than error (such as the FATAL level), then log4jdbc will not log anything and in fact the actual (real) connection to the underlying database will be returned by the log4jdbc driver (thus allowing log4jdbc to be installed and available to turn on at runtime at a moment's notice without imposing any actual performance loss when not being used). 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.

Each of these logs can be set at either DEBUG, INFO or ERROR level.

- **DEBUG** includes the class name and line number (if available) at which the SQL was executed. **Use DEBUG level with extra care, as this imposes an additional performance penalty when in use.**
- **INFO** includes the SQL (or other information as applicable.)
- **ERROR** will show the stack traces in the log output when SQLExceptions occur.

logger	description	since
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.	1.0
jdbc.sqltiming	Logs the SQL, post-execution, including timing statistics on how long the SQL took to execute.	1.0
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.	1.0
jdbc.resultset	Even more voluminous, because all calls to ResultSet objects are logged.	1.0
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.	1.2alpha1

Additionally, there is one logger named **log4jdbc.debug** which is for use with internal debugging of log4jdbc. At this time it just prints out information on which underlying drivers were found and not found when the log4jdbc spy driver loads.

In a typical usage scenario, you might turn on only the jdbc.sqlonly logging at INFO level, just to view the SQL coming out of your program.

Then if you wanted to view how long each SQL statement is taking to execute, you might use jdbc.sqltiming.

jdbc.audit, jdbc.resultset and jdbc.connection are used for more in depth diagnosis of what is going on under the hood with JDBC as potentially almost every single call to JDBC could be logged (logs can grow very large, very quickly with jdbc.audit and jdbc.resultset!)

Because SLF4J can be used with many popular java logging systems, the setup for your loggers will vary depending on which underlying logging system you use. Sample configuration files for log4j are provided here:

log4j.xml and log4j.properties.

6. Adjust debugging options.

When logging at the DEBUG level, the class file and line number (if available) for the class that invoked JDBC is logged after each log statement. This is enormously useful for finding where in the code the SQL is generated. Be careful when using this on a production system because there is a small, but potentially significant penalty performance for generating this data on each logging statement.

In many cases, this call stack data is not very useful because the calling class into log4jdbc is a connection pool, object-persistance layer or other layer between log4jdbc and your application code -- but the class file and line number information you really are interested in seeing is where in your application the SQL was generated from.

Set the log4jdbc.debug.stack.prefix System property for log4jdc to help get around this problem:

```
-Dlog4jdbc.debug.stack.prefix=<package.prefix>
```

Where <package.prefix> is a String that is the partial (or full) package prefix for the package name of your application. The call stack will be searched down to the first occurence of a class that has the matching prefix. If this is not set, the actual class that called into log4jdbc is used in the debug output (in many cases this will be a connection pool class)

For example, setting a system property such as this:

```
-Dlog4jdbc.debug.stack.prefix=com.mycompany.myapp
```

Would cause the call stack to be searched for the first call that came from code in the com.mycompany.myapp package or below, thus if all of your SQL generating code was in code located in the com.mycompany.myapp package or any subpackages, this would be printed in the debug information, rather than the package name for a connection pool, object relational system, etc.

Options

log4jdbc options are controlled via system properties. The simplest way to set these is with the java -D command line option. For example:

```
java -Dlog4jdbc.drivers=my.funky.DriverClass -classpath ./classes my.funky.Program
```

system property	default	description	sin
log4jdbc.drivers		One or more fully qualified class names for JDBC drivers that log4jdbc should load and wrap. If more than one driver needs to be specified here, they should be comma separated with no spaces. This option is not normally needed because most popular JDBC drivers are already loaded by default-- this should be used if one or more additional JDBC drivers that (log4jdbc doesn't already wrap) needs to be included.	1.0
log4jdbc.debug.stack.prefix		The partial (or full) package prefix for the package name of your application. The call stack will be searched down to the first occurrence of a class that has the matching prefix. If this is not set, the actual class that called into log4jdbc is used in the debug output (in many cases this will be a connection pool class.) For example, setting a system property such as this: -Dlog4jdbc.debug.stack.prefix=com.mycompany.myapp Would cause the call stack to be searched for the first call that came from code in the com.mycompany.myapp package or below, thus if all of your sql generating code was in code located in the com.mycompany.myapp package or any subpackages, this would be printed in the debug information, rather than the package name for a connection pool, object relational system, etc.	1.0

log4jdbc.sqltiming.warn.threshold		Millisecond time value. Causes SQL that takes the number of milliseconds specified or more time to execute to be logged at the warning level in the sqltiming log. Note that the sqltiming log must be enabled at the warn log level for this feature to work. Also the logged output for this setting will log with debug information that is normally only shown when the sqltiming log is enabled at the debug level. This can help you to more quickly find slower running SQL without adding overhead or logging for normal running SQL that executes below the threshold level (if the logging level is set appropriately.)	1.1
log4jdbc.sqltiming.error.threshold		Millisecond time value. Causes SQL that takes the number of milliseconds specified or more time to execute to be logged at the error level in the sqltiming log. Note that the sqltiming log must be enabled at the error log level for this feature to work. Also the logged output for this setting will log with debug information that is normally only shown when the sqltiming log is enabled at the debug level. This can help you to more quickly find slower running SQL without adding overhead or logging for normal running SQL that executes below the threshold level (if the logging level is set appropriately.)	1.1
log4jdbc.dump.booleanastruefalse	false	When dumping boolean values in SQL, dump them as 'true' or 'false'. If this option is not set, they will be dumped as 1 or 0 as many databases do not have a boolean type, and this allows for more portable sql dumping.	1.2
log4jdbc.dump.sql.maxlinelength	90	When dumping SQL, if this is greater than 0, than the dumped SQL will be broken up into lines that are no longer than this value. Set this value to 0 if you don't want log4jdbc to try and break the SQL into lines this way. In future versions of log4jdbc, this will probably default to 0.	1.2
log4jdbc.dump.fulldebugstacktrace	false	If dumping in debug mode, dump the full stack trace. This will result in EXTREMELY voluminous output, but can be very useful under some circumstances when trying to track down the call chain for generated SQL.	1.2
log4jdbc.dump.sql.select	true	Set this to false to suppress SQL select statements in the output.	1.2
log4jdbc.dump.sql.insert	true	Set this to false to suppress SQL insert statements in the output.	1.2
log4jdbc.dump.sql.update	true	Set this to false to suppress SQL update statements in the output.	1.2
log4jdbc.dump.sql.delete	true	Set this to false to suppress SQL delete statements in the output.	1.2
log4jdbc.dump.sql.create	true	Set this to false to suppress SQL create statements in the output.	1.2
log4jdbc.dump.sql.addsemicolon	false	Set this to true to add an extra semicolon to the end of SQL in the output. This can be useful when you want to generate SQL from a program with log4jdbc in order to create a script to feed back into a database to run at a later time.	1.2

log4jdbc.statement.warn	false	Set this to true to display warnings (Why would you care?) in the log when Statements are used in the log. NOTE, this was always true in releases previous to 1.2alpha2. It is false by default starting with release 1.2 alpha 2.	1.2

Other

A simple tool is included which you can use to post-process sql timing logs produced by log4jdbc. It can output simple profiling reports with statistics and a dump of the sql statements that ran the slowest within the log. To invoke the tool, use **profsql.sh** (for unix/linux) and **profsql.cmd** (for windows) located in the scripts folder. These scripts take as one argument, the filename of a sql timing log (generated from the jdbc.sqltiming log category. They produce a profiling report to stdout. The tool is currently experimental and I expect it to evolve quite a bit over the next few releases. Nevertheless, it has already been very useful to me for tracking down SQL performance problems.

Similiar Tools

Some other tools and libraries that are similiar to log4jdbc.

- P6Spy is probably the most well known JDBC logging driver but it hasn't been updated in over 5 years.
- Craftsman Spy appears to overlap quite a bit with the feature set in log4jdbc. This library hasn't been updated in 2 years and depends on Jakarta Commons Logging.
- JAMon (Java Application Monitor) is a comprehensive application monitor and monitoring API which includes JDBC/SQL monitoring as part of it's very large feature set.
- JdbcProxy The driver can also emulate another JDBC driver to test the application without a database.
- LogDriver appears to be similiar to log4jdbc and the author has written a nice article on JDBC logging in general and his motivation and experience of writing LogDriver.
- and yet another JDBC logger