OJB logging configuration

by Thomas Dudziak

Table of contents

1. Logging in OJB

For generating log messages, OJB provides its own, simplistic logging component PoorMansLoggerImpl, but is also able to use the two most common Java logging libraries, commons-logging (which is actually a wrapper around several logging components) and Log4j. In addition, it is also possible to define your own logging implementation.

Per default, OJB uses its own PoorMansLoggerImpl which does not require configuration and prints to `stdout`.

2. Logging configuration within OJB

2.1. How and when OJB determines what kind of logging to use

Logging is the first component of OJB that is initialized. If you access any component of OJB, logging will be initialized first before that component is doing anything else. Therefore, you'll have to provide for the configuration of logging before you access OJB in your program (this is mostly relevant if you plan to initialize OJB at runtime as is described below). Please note that logging configuration is independent of the configuration of other parts of OJB, namely the runtime (via OJB.properties) and the database/repository (via repository.xml).

These are the individual steps OJB performs in order to initialize the logging component:

1. First, OJB checks whether the system property `org.apache.ojb.broker.util.logging.Logger.class` is set. If specified, this property gives the fully qualified class name of the logger class (a class implementing the Logger interface). Along with this property, another property is then read which may specify a properties file for this logger class, `org.apache.ojb.broker.util.logging.Logger.configFile`.
2. If this property is not set, then OJB tries to read the file `OJB-logging.properties`. The name and path of this file can be changed by setting the runtime property of the same name. See below for the contents of this file.
3. For backwards compatibility, OJB next tries to read the logging settings from the file OJB.properties which is the normal runtime configuration file of OJB. Again, the name and path of this file can be changed by setting the runtime property of the same name. This file may contain the same entries as the `OJB-logging.properties` file.
4. If the the `OJB.properties` file does not contain logging settings, next it is checked whether the commons-logging log property `org.apache.commons.logging.Log` or the commons-logging log factory system property `org.apache.commons.logging.LogFactory` is set. If that's the case, OJB will use commons-logging for its logging purposes.
5. Next, OJB checks for the presence of the Log4j properties file `log4j.properties`. If it is found, the OJB uses Log4j directley (without commons-logging).
6. Finally, OJB tries to find the commons-logging properties file `commons-logging.properties` which when found directs OJB to use commons-logging for its logging.
7. If none of the above is true, or if the specified logger class could not be found or initialized, then OJB defaults to its `PoorMansLoggerImpl` logger which simply logs to `stdout`.

The only OJB component whose logging is not initialized this way, is the boot logger which is used by logging component itself and a few other core components. It will (for obvious reasons) always use PoorMansLoggerImpl and therefore log to `stdout`. You can define the log level of the boot logger via the `OJB.bootLogLevel` system property. Per default, **WARN** is used.

2.2. Configuration of logging for the individual components

Regardless of the logging implementation that is used by OJB, the configuration is generally similar. The individual logging implementations mainly differ in the syntax and in the configuration of the format of the output and of the output target (where to log to). See below for specific details and examples.
In general, you specify a default log level and for every component (usually a class) that should log differently, the amount and level of detail that is logged about that component. These are the levels:

DEBUG
Messages that express what OJB is currently doing. This is the most detailed debugging level
INFO
Informational messages
WARN
Warnings that may denote potentional problems (this is the default level)
ERROR
As the name says, this level is for errors which means that some action could not be completed successfully
FATAL
Fatal errors which usually prevent an application from continuing

The levels **DEBUG** and **INFO** usually result in a lot of log messages which will reduce the performance of the application. Therefore these levels should only be used when necessary.

There are two special loggers to be aware of. The **boot logger** is the logger used by the logging component itself as well as a few other core components. It will therefore always use the PoorMansLoggerImpl logging implementation. You can configure its logging level via the `OJB.bootLogLevel` system property.
The **default logger** is denoted in the `OJB-logging.properties` file by the keyword `DEFAULT` instead of the class name. It is used by components that don't require their own logging configuration (usually because they are rather small components).

3. Logging configuration via configuration files

3.1. OJB-logging.properties

This file usually specifies which logging implementation to use using the `org.apache.ojb.broker.util.logging.Logger.class` property, and which properties file this logger has (if any) using the `org.apache.ojb.broker.util.logging.Logger.configFile` property. You should also use this file to specify log levels for OJB's components if you're not using Log4j or commons-logging (which have their own configuration files).

A typical `OJB-logging.properties` file looks like this:

```
# Which logger to use
org.apache.ojb.broker.util.logging.Logger.class=org.apache.ojb.broker.util.logging.PoorMar

# Configuration file of the logger
#org.apache.ojb.broker.util.logging.Logger.configFile=

# Global default log level used for all logging entities if not specified
ROOT.LogLevel=ERROR

# The log level of the default logger
DEFAULT.LogLevel=WARN
```

```
# Logger for PersistenceBrokerImpl class
org.apache.ojb.broker.core.PersistenceBrokerImpl.LogLevel=WARN

# Logger for RepositoryXmlHandler, useful for debugging parsing of
repository.xml!
org.apache.ojb.broker.metadata.RepositoryXmlHandler.LogLevel=WARN
```

3.2. commons-logging.properties

This file is used by commons-logging. For details on its structure see here.

An example `commons-logging.properties` file would be:

```
# Use Log4j
org.apache.commons.logging.Log=org.apache.commons.logging.impl.Log4JLogger

# Configuration file of the log
log4j.configuration=log4j.properties
```

Note:

Since commons-logging provides the same function as the logging component of OJB, it will likely be used as OJB's logging component in the near future.

3.3. log4j.properties

The commons-logging configuration file. Details can be found here.

A sample log4j configuration is:

```
# Root logging level is WARN, and we're using two logging targets
log4j.rootCategory=WARN, A1, A2

# A1 is set to be ConsoleAppender sending its output to System.out
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-5r %-5p [%t] %c{2} - %m%n

# Appender A2 writes to the file "org.apache.ojb.log".
log4j.appender.A2=org.apache.log4j.FileAppender
log4j.appender.A2.File=org.apache.ojb.log

# Truncate the log file if it aleady exists.
log4j.appender.A2.Append=false

# A2 uses the PatternLayout.
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%-5r %-5p [%t] %c{2} - %m%n

# Special logging directives for individual components
log4j.logger.org.apache.ojb.broker.metadata.RepositoryXmlHandler=DEBUG
log4j.logger.org.apache.ojb.broker.accesslayer.ConnectionManager=INFO
log4j.logger.org.apache.ojb.odmg=INFO
```

3.4. Where to put the configuration files

OJB and the different logging implementations usually look up their configuration files in the classpath. So for instance, OJB searches for the `OJB-logging.properties` file directly in any of the entries of the classpath, directories and jar files. If the classpath contains in that order `some-library.jar`, `db-ojb.jar`, and `.`, then it will first search in the two jars (which themselves contain a directory structure in which OJB will search only in the root), and lastly in the current directory (which only happens if `.` is part of the classpath) but not in sub directories of it.

For applications, this classpath can easily be set either as an environment variable `CLASSPATH` or by using the commandline switch `-classpath` when invoking the java executable.

For web applications however, the server will define the classpath. There are specific folders in the webapp structure that are always part of the webapp's classpath. The one that is normally used to store configuration files, is the `classes` folder:

```
[folder containing webapps]\
    mywebapp\
        WEB-INF\
            lib\
            classes\   <-- Put your configuration files here
```

4. Logging configuration at runtime

Sometimes you want to configure OJB completely at runtime (within your program). How to do that for logging depends on the used logging implementation, but you can usually configure them via system properties. The only thing to keep in mind is that logging in OJB is initialized as soon as you use one of its components, so you'll have to define the properties prior to using any OJB parts.

With system properties (which are accessible via `System.getProperty()` from within a Java program) you can always define the following OJB logging settings:

`org.apache.ojb.broker.util.logging.Logger.class`
Which logger OJB shall use
`org.apache.ojb.broker.util.logging.Logger.configFile`
The config file of the logger
`OJB-logging.properties`
The path to the logging properties file, default is `OJB-logging.properties`
`OJB.properties`
The path to the OJB properties file (which may contain logging settings), default is `OJB.properties`
`org.apache.commons.logging.Log`
Use commons-logging with the specified log implementation
`org.apache.commons.logging.LogFactory`
Use commons-logging with the specified log factory
`log4j.configuration`
When using Log4j directly or via commons-logging, this is the Log4j configuration file (default is `log4j.properties`)

In addition, all Log4j properties (e.g. `log4j.rootCategory`) can be specified as system properties.

5. Defining your own logger

It is rather easy to use your own logger. All you need to do is to provide a class that implements the interface Logger. Besides the actual log methods (`debug, info, warn, error, fatal`) this interface defines a method `void configure(Configuration)` which is used to initialize the logger with the logging properties (as contained in `OJB-logging.properties`).

Note:
Because commons-logging performs a similar function to the OJB logging component, it is likely that it will be used as such in the near future. Therefore you're encouraged to also implement the Log interface which is nearly the same as the Logger interface.