Logging

Table of contents

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

1 Introduction

The Grinder 3.7 replaced a previous custom logging framework with Logback (http://logback.qos.ch/) . Scripts now use a standard logging API (SLF4J (http://www.slf4j.org/)), and Logback can be configured to alter the output format, manage archiving of log files, and direct log streams to alternative locations.

2 Changing the Logback configuration

The Grinder uses two Logback configuration files:

- `logback.xml` - Used by all processes. Logs to the terminal (`stdout`, `stderr`).
- `logback-worker.xml` - Used by worker processes. Configures logging to the log file and the data log file.

Both configuration files are located in the `grinder-core.jar` file. Refer to the Logback manual (http://logback.qos.ch/manual/index.html) for full details of the configuration file settings.

Let's change the archive settings for the output log to keep more than one archive file. First, extract the configuration file.

```
cd lib
jar xf grinder-core-3.7.jar logback-worker.xml
```

Open the `logback-worker.xml` file in a text editor and locate the `log-file` appender. To keep five archive files, simply change the `maxIndex` setting to`5` so it matches the following:

```
<appender name="log-file"
  class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${PREFIX}.log</file>

  <encoder>
    <pattern>%d %-5level %logger{0} %marker: %message%n</pattern>
  </encoder>

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
    <fileNamePattern>${PREFIX}.log%i</fileNamePattern>
    <minIndex>1</minIndex>
    <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="net.grinder.util.logback.RollOnStartUp" />

</appender>
```

Save the file under the same name (`logback-worker.xml`). To use the modified configuration, add the file's directory to the `CLASSPATH` used to start The Grinder. We extracted the file into the `lib` directory, so we could start the agent process with something like the following:

```
cd $GRINDER_HOME
java -classpath lib:lib/grinder.jar net.grinder.Grinder
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

3 Logging data to a database

The `logback-worker.xml` file configures two Logback loggers: `worker` for the main log file, and `data` for the data log file. Let's change the configuration to send test data to a database. To do this, we'll configure a new appender and add it to the data logger. Logback's database appender supports several databases; here's a suitable configuration for an Oracle database.

```
<appender name="data-db" class="ch.qos.logback.classic.db.DBAppender">
  <connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">
    <driverClass>oracle.jdbc.OracleDriver</driverClass>
    <url>jdbc:oracle:thin:@localhost:1521:XE</url>
    <user>grinder</user>
    <password>grinder</password>
  </connectionSource>
</appender>

<logger name="data" additivity="false">
  <appender-ref ref="data-file" />
  <appender-ref ref="data-db" />
</logger>
```

To expose any problems with the configuration, we'll also enable the Logback debug output by changing the first line of the configuration.

```
 <configuration debug="true">
```

Before we can use the database appender, we need to set up the appropriate database tables. To do this, create a suitable database account (the configuration above uses an account called *grinder*), download the Logback distribution, and locate and execute the appropriate DDL (`logback-classic/src/main/java/ch/qos/logback/classic/db/dialect/oracle.sql` for Oracle).

To run the configuration, add the directory containing `logback-worker.xml` to the `CLASSPATH`, along with the appropriate database driver. For example:

```
java -classpath /usr/lib/oracle/xe/app/oracle/product/10.2.0/server/jdbc/lib/
ojdbc14.jar:lib:lib/grinder.jar  net.grinder.Grinder
```

4 Writing a custom appender for data logs

If you tried out the database configuration in the previous section you may have noted the following drawbacks.

- It's not particularly fast. Maximum logging throughput is of the order of a hundred log events per second, and this severely constrains the rate at which a worker process can perform tests.
- The log data is written as a string to a single `formatted_message` column. This is not amenable to further processing.

To address these problems, you will have to write a custom database appender, perhaps by modifying the standard Logback-supplied appender. If you write such an appender, please consider making it generic and contributing it back to The Grinder. The following sections contain some implementation ideas.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

4.1 Improving database logging performance

The most beneficial change from a performance perspective would be to buffer the log events, and write many events to the database at once. JDBC batching would further improve performance. I suspect that this change alone would allow tens of thousands of events to be logged per second.

The standard appender includes caller data (filename, class, method, line) that is expensive to obtain and is of little use for The Grinder data log. It also logs exception and property information. These features can be removed.

To support the secondary exception and property tables, the standard appender needs to obtain the primary key of the newly logged event. Unfortunately this uses an appender level lock (unnecessarily, it could have synchronised on the database connection instead), and becomes a bottleneck when many worker threads are using the appender. Since the exception and property tables are unnecessary, this complexity can also be removed.

4.2 Customising data log output

The Grinder data logger generates `ILoggingEvent`s with the formatted string set to a comma-separated string (formatted as in the standard data log). It also supplies an instance of `net.grinder.engine.process.DataLogArguments` as the first and only argument. This can be accessed using `ILoggingEvent.getArgumentArray()[0]`.

The `DataLogArguments` object provides all the information you might need about a particular data log event, including the thread and run numbers, the `Test`, and the raw statistics. Refer to the `net.grinder.engine.processs.ThreadDataLogger` source code for an example of how to extract the appropriate statistics values from the raw statistics.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.