Reasons to prefer logback over log4j

Logback brings a very large number of improvements over log4j, big and small. They are too many to enumerate exhaustively. Nevertheless, here is a non-exhaustive list of reasons for switching to logback from log4j. Keep in mind that logback is conceptually very similar to log4j as both projects were founded by the same developer. If you are already familiar with log4j, you will quickly feel at home using logback. If you like log4j, you will probably love logback.

Faster implementation 快

Based on our previous work on log4j, logback internals have been re-written to perform about ten times faster on certain critical execution paths. Not only are logback components faster, they have a smaller memory footprint as well.

Extensive battery of tests 广泛的测试用例

Logback comes with a very extensive battery of tests developed over the course of several years and untold hours of work. While log4j is also tested, logback takes testing to a completely different level. In our opinion, this is the single most important reason to prefer logback over log4j. You want your logging framework to be rock solid and dependable even under adverse conditions.

logback-classic speaks SLF4J natively

Since the `Logger` class in logback-classic implements the SLF4J API natively, you incur zero overhead when invoking an SLF4J logger with logback-classic as the underlying implementation. Moreover, since logback-classic strongly encourages the use of SLF4J as its client API, if you need to switch to log4j or to j.u.l., you can do so by replacing one jar file with another. You will not need to touch your code logging via the SLF4J API. This can drastically reduce the work involved in switching logging frameworks.

Extensive documentation 广泛的文档

Logback ships with detailed and constantly updated documentation.

Configuration files in XML or Groovy 配置文件

The traditional way of configuring logback is via an XML file. Most of the examples in the documentation use this XML syntax. However, as of logback version 0.9.22, configuration files written in Groovy are also supported. Compared to XML, Groovy-style configuration is more intuitive, consistent and has a shorter syntax.

There is also a tool to automatically migrate your logback.xml files to logback.groovy.

Automatic reloading of configuration files 自动重新加载配置文件

Logback-classic can automatically reload its configuration file upon modification. The scanning process is fast, contention-free, and dynamically scales to millions of invocations per second spread over hundreds of threads. It also plays well within application servers and more generally within the JEE environment as it does *not* involve the creation of a separate thread for scanning.

Graceful recovery from I/O failures 优雅地从I/O故障中恢复

Logback's `FileAppender` and all its sub-classes, including `RollingFileAppender`, can gracefully recover from I/O failures. Thus, if a file server fails temporarily, you no longer need to restart your application just to get logging working again. As soon as the file server comes back up, the relevant logback appender will transparently and quickly recover from the previous error condition.

Automatic removal of old log archives 旧的日志档案自动地删除

By setting the maxHistory property of TimeBasedRollingPolicy or SizeAndTimeBasedFNATP, you can control the maximum number of archived files. If your rolling policy calls for monthly rollover and you wish to keep one year's worth of logs, simply set the maxHistory property to 12. Archived log files older than 12 months will be automatically removed.

Automatic compression of archived log files 存档的日志文件的自动压缩

RollingFileAppender can automatically compress archived log files during rollover. Compression always occurs asynchronously so that even for large log files, your application is not blocked for the duration of the compression. 压缩总是异步地发生

Prudent mode

In prudent mode, multiple `FileAppender` instances running on multiple JVMs can safely write to the same log file. With certain limitations, prudent mode extends to `RollingFileAppender`.

Lilith

Lilith is a logging and access event viewer for logback. It is comparable to log4j's chainsaw, except that Lilith is designed to handle large amounts of logging data without flinching.

Conditional processing of configuration files 配置文件的条件处理

Developers often need to juggle between several logback configuration files targeting different environments such as development, testing and production. These configuration files have substantial parts in common, differing only in a few places. To avoid duplication, logback supports conditional processing of configuration files with the help of `<if>`, `<then>` and `<else>` elements so that a single configuration file can adequately target several environments.

Filters 过滤器

Logback comes with a wide array of filtering capabilities going much further than what log4j has to offer. For example, let's assume that you have a business-critical application deployed on a production server. Given the large volume of transactions processed, logging level is set to WARN so that only warnings and errors are logged. Now imagine that you are confronted with a bug that can be reproduced on the production system but remains elusive on the test platform due to unspecified differences between those two environments (production/testing).

With log4j, your only choice is to lower the logging level to DEBUG on the production system in an attempt to identify the problem. Unfortunately, this will generate large volume of logging data, making analysis difficult. More importantly, extensive logging can impact the performance of your

application on the production system.

With logback, you have the option of keeping logging at the WARN level for all users except for the one user, say Alice, who is responsible for identifying the problem. When Alice is logged on, she will be logging at level DEBUG while other users can continue to log at the WARN level. This feat can be accomplished by adding 4 lines of XML to your configuration file. Search for `MDCFilter` in the relevant section of the manual.

SiftingAppender

SiftingAppender is an amazingly versatile appender. It can be used to separate (or sift) logging according to **any** given runtime attribute. For example, `SiftingAppender` can separate logging events according to user sessions, so that the logs generated by each user go into distinct log files, one log file per user.

Stack traces with packaging data

When logback prints an exception, the stack trace will include packaging data. Here is a sample stack trace generated by the logback-demo web-application.

```
14:28:48.835 [btpool0-7] INFO  c.q.l.demo.prime.PrimeAction - 99 is not a valid value
java.lang.Exception: 99 is invalid
  at ch.qos.logback.demo.prime.PrimeAction.execute(PrimeAction.java:28) [classes/:na]
  at org.apache.struts.action.RequestProcessor.processActionPerform(RequestProcessor.java:431) [struts-1.2.9.jar:1.2.9]
  at org.apache.struts.action.RequestProcessor.process(RequestProcessor.java:236) [struts-1.2.9.jar:1.2.9]
  at org.apache.struts.action.ActionServlet.doPost(ActionServlet.java:432) [struts-1.2.9.jar:1.2.9]
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:820) [servlet-api-2.5-6.1.12.jar:6.1.12]
  at org.mortbay.jetty.servlet.ServletHolder.handle(ServletHolder.java:502) [jetty-6.1.12.jar:6.1.12]
  at ch.qos.logback.demo.UserServletFilter.doFilter(UserServletFilter.java:44) [classes/:na]
  at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1115) [jetty-6.1.12.jar:6.1.12]
  at org.mortbay.jetty.servlet.ServletHandler.handle(ServletHandler.java:361) [jetty-6.1.12.jar:6.1.12]
  at org.mortbay.jetty.webapp.WebAppContext.handle(WebAppContext.java:417) [jetty-6.1.12.jar:6.1.12]
  at org.mortbay.jetty.handler.ContextHandlerCollection.handle(ContextHandlerCollection.java:230) [jetty-6.1.12.jar:6.1.12]
```

From the above, you can recognize that the application is using Struts version 1.2.9 and was deployed under jetty version 6.1.12. Thus, stack traces will quickly inform the reader about the classes intervening in the exception but also the package and package versions they belong to. When your customers send you a stack trace, as a developer you will no longer need to ask them to send you information about the versions of packages they are using. The information will be part of the stack trace. See "%xThrowable" conversion word for details.

This feature can be quite helpful to the point that some users mistakenly consider it a feature of their IDE.

Logback-access, i.e. HTTP-access logging with brains, is an integral part of logback

Last but not least, the logback-access module, part of the logback distribution, integrates with Servlet containers such as Jetty or Tomcat to provide rich and powerful HTTP-access log functionality. Since logback-access was part of the initial design, all the logback-classic features you love are available in logback-access as well.

In summary 总结

We have listed a number of reasons for preferring logback over log4j. Given that logback builds upon on our previous work on log4j, simply put, logback is just a better log4j.