Log4j

log4j is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License.

log4j has been ported to the C, C++, C#, Perl, Python, Ruby, and Eiffel languages.

log4j is highly configurable through external configuration files at runtime. It views the logging process in terms of levels of priorities and offers mechanisms to direct logging information to a great variety of destinations, such as a database, file, console, UNIX Syslog, etc.

log4j has three main components:

* **loggers**: Responsible for capturing logging information.
* **appenders**: Responsible for publishing logging information to various preferred destinations.
* **layouts**: Responsible for formatting logging information in different styles.

History of log4j

* Started in early 1996 as tracing API for the E.U. SEMPER (Secure Electronic Marketplace for Europe) project.
* After countless enhancements and several incarnations, the initial API has evolved to become log4j, a popular logging package for Java.
* The package is distributed under the Apache Software License, a full-fledged open source license certified by the open source initiative.
* The latest log4j version, including its full-source code, class files, and documentation can be found at <http://logging.apache.org/log4j/>.

log4j Features

* It is thread-safe.
* It is optimized for speed.
* It is based on a named logger hierarchy.
* It supports multiple output appenders per logger.
* It supports internationalization.
* It is not restricted to a predefined set of facilities.
* Logging behavior can be set at runtime using a configuration file.
* It is designed to handle Java Exceptions from the start.
* It uses multiple levels, namely ALL, TRACE, DEBUG, INFO, WARN, ERROR and FATAL.
* The format of the log output can be easily changed by extending the *Layout* class.
* The target of the log output as well as the writing strategy can be altered by implementations of the Appender interface.
* It is fail-stop. However, although it certainly strives to ensure delivery, log4j does not guarantee that each log statement will be delivered to its destination.

Pros and Cons of Logging

Logging is an important component of the software development. A well-written logging code offers quick debugging, easy maintenance, and structured storage of an application's runtime information.

Logging does have its drawbacks also. It can slow down an application. If too verbose, it can cause scrolling blindness. To alleviate these concerns, log4j is designed to be reliable, fast and extensible.

Since logging is rarely the main focus of an application, the log4j API strives to be simple to understand and to use.

### Level Object

The Level object defines the granularity and priority of any logging information. There are seven levels of logging defined within the API: OFF, DEBUG, INFO, ERROR, WARN, FATAL, and ALL.

### Filter Object

The Filter object is used to analyze logging information and make further decisions on whether that information should be logged or not.

An Appender objects can have several Filter objects associated with them. If logging information is passed to a particular Appender object, all the Filter objects associated with that Appender need to approve the logging information before it can be published to the attached destination.

### ObjectRenderer

The ObjectRenderer object is specialized in providing a String representation of different objects passed to the logging framework. This object is used by Layout objects to prepare the final logging information.

### LogManager

The LogManager object manages the logging framework. It is responsible for reading the initial configuration parameters from a system-wide configuration file or a configuration class.

|  |  |
| --- | --- |
| **Property** | **Description** |
| layout | Appender uses the Layout objects and the conversion pattern associated with them to format the logging information. |
| target | The target may be a console, a file, or another item depending on the appender. |
| level | The level is required to control the filtration of the log messages. |
| threshold | Appender can have a threshold level associated with it independent of the logger level. The Appender ignores any logging messages that have a level lower than the threshold level. |
| filter | The Filter objects can analyze logging information beyond level matching and decide whether logging requests should be handled by a particular Appender or ignored. |

The addAppender() method adds an Appender to the Logger object. As the example configuration demonstrates, it is possible to add many Appender objects to a logger in a comma-separated list, each printing logging information to separate destinations.

We have used only one appender *FileAppender* in our example above. All the possible appender options are:

* AppenderSkeleton
* AsyncAppender
* ConsoleAppender
* DailyRollingFileAppender
* ExternallyRolledFileAppender
* FileAppender
* JDBCAppender
* JMSAppender
* LF5Appender
* NTEventLogAppender
* NullAppender
* RollingFileAppender
* SMTPAppender
* SocketAppender
* SocketHubAppender
* SyslogAppender
* TelnetAppender
* WriterAppender

We would cover FileAppender in [Logging in Files](https://www.tutorialspoint.com/log4j/log4j_logging_files.htm) and JDBC Appender would be covered in [Logging in Database](https://www.tutorialspoint.com/log4j/log4j_logging_database.htm).

Layout

We have used PatternLayout with our appender. All the possible options are:

* DateLayout
* HTMLLayout
* PatternLayout
* SimpleLayout
* XMLLayout
* Logging Methods
* Once we obtain an instance of a named logger, we can use several methods of the logger to log messages. The Logger class has the following methods for printing the logging information.

|  |  |
| --- | --- |
| **#** | **Methods and Description** |
| 1 | **public void debug(Object message)**  It prints messages with the level Level.DEBUG. |
| 2 | **public void error(Object message)**  It prints messages with the level Level.ERROR. |
| 3 | **public void fatal(Object message)**  It prints messages with the level Level.FATAL. |
| 4 | **public void info(Object message)**  It prints messages with the level Level.INFO. |
| 5 | **public void warn(Object message)**  It prints messages with the level Level.WARN. |
| 6 | **public void trace(Object message)**  It prints messages with the level Level.TRACE. |

Following is a sample configuration file **log4j.properties** for FileAppender −

# Define the root logger with appender file

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.FileAppender

# Set the name of the file

log4j.appender.FILE.File=${log}/log.out

# Set the immediate flush to true (default)

log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode

log4j.appender.FILE.Threshold=debug

# Set the append to false, overwrite

log4j.appender.FILE.Append=false

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%m%n

If you wish to have an XML configuration file equivalent to the above **log4j.properties** file, then here is the content:

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration>

<appender name="FILE" class="org.apache.log4j.FileAppender">

<param name="file" value="${log}/log.out"/>

<param name="immediateFlush" value="true"/>

<param name="threshold" value="debug"/>

<param name="append" value="false"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="conversionPattern" value="%m%n"/>

</layout>

</appender>

<logger name="log4j.rootLogger" additivity="false">

<level value="DEBUG"/>

<appender-ref ref="FILE"/>

</logger>

</log4j:configuration>

You can try [log4j - Sample Program](http://www.tutorialspoint.com/log4j/log4j_sample_program.htm) with the above configuration.

Logging in Multiple Files

You may want to write your log messages into multiple files for certain reasons, for example, if the file size reached to a certain threshold.

To write your logging information into multiple files, you would have to use **org.apache.log4j.RollingFileAppender** class which extends the **FileAppender** class and inherits all its properties.

We have the following configurable parameters in addition to the ones mentioned above for FileAppender −

|  |  |
| --- | --- |
| **Property** | **Description** |
| maxFileSize | This is the critical size of the file above which the file will be rolled. Default value is 10 MB. |
| maxBackupIndex | This property denotes the number of backup files to be created. Default value is 1. |

Following is a sample configuration file **log4j.properties** for RollingFileAppender.

# Define the root logger with appender file

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.RollingFileAppender

# Set the name of the file

log4j.appender.FILE.File=${log}/log.out

# Set the immediate flush to true (default)

log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode

log4j.appender.FILE.Threshold=debug

# Set the append to false, should not overwrite

log4j.appender.FILE.Append=true

# Set the maximum file size before rollover

log4j.appender.FILE.MaxFileSize=5KB

# Set the the backup index

log4j.appender.FILE.MaxBackupIndex=2

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%m%n

If you wish to have an XML configuration file, you can generate the same as mentioned in the initial section and add only additional parameters related to **RollingFileAppender**.

This example configuration demonstrates that the maximum permissible size of each log file is 5 MB. Upon exceeding the maximum size, a new log file will be created. Since **maxBackupIndex** is defined as 2, once the second log file reaches the maximum size, the first log file will be erased and thereafter, all the logging information will be rolled back to the first log file.

You can try [log4j - Sample Program](http://www.tutorialspoint.com/log4j/log4j_sample_program.htm) with the above configuration.

Daily Log File Generation

There may be a requirement to generate your log files on a daily basis to keep a clean record of your logging information.

To write your logging information into files on a daily basis, you would have to use **org.apache.log4j.DailyRollingFileAppender** class which extends the **FileAppender** class and inherits all its properties.

There is only one important configurable parameter in addition to the ones mentioned above for FileAppender:

|  |  |
| --- | --- |
| **Property** | **Description** |
| DatePattern | This indicates when to roll over the file and the naming convention to be followed. By default, roll over is performed at midnight each day. |

DatePattern controls the rollover schedule using one of the following patterns:

|  |  |
| --- | --- |
| **DatePattern** | **Description** |
| '.' yyyy-MM | Roll over at the end of each month and at the beginning of the next month. |
| '.' yyyy-MM-dd | Roll over at midnight each day. This is the default value. |
| '.' yyyy-MM-dd-a | Roll over at midday and midnight of each day. |
| '.' yyyy-MM-dd-HH | Roll over at the top of every hour. |
| '.' yyyy-MM-dd-HH-mm | Roll over every minute. |
| '.' yyyy-ww | Roll over on the first day of each week depending upon the locale. |

Following is a sample configuration file **log4j.properties** to generate log files rolling over at midday and midnight of each day.

# Define the root logger with appender file

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.DailyRollingFileAppender

# Set the name of the file

log4j.appender.FILE.File=${log}/log.out

# Set the immediate flush to true (default)

log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode

log4j.appender.FILE.Threshold=debug

# Set the append to false, should not overwrite

log4j.appender.FILE.Append=true

# Set the DatePattern

log4j.appender.FILE.DatePattern='.' yyyy-MM-dd-a

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%m%n