LOG4J

APACHE LOG4J

asynchronous javascript and xml

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Log4j is a popular logging package written in Java. Log4J is ported to the C, C++, C#, Perl, Python, Ruby, and Eiffel languages.

Audience

This tutorial is prepared for beginners to help them understand the basic functionality of Log4J logging framework.

Prerequisites

As you are going to use Log4J logging framework in various Java-based application development, it is imperative that you should have a good understanding of Java programming language.

Copyright & Disclaimer

Table of Contents

1. OVERVIEW

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.

2. INSTALLATION

Log4j API 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/.

To install log4j on your system, download apache-log4j-x.x.x.tar.gz from the specified URL and follow the steps given below.

Step 1

Unzip and untar the downloaded file in /usr/local/ directory as follows:

```
$ gunzip apache-log4j-1.2.15.tar.gz
$ tar -xvf apache-log4j-1.2.15.tar
apache-log4j-1.2.15/tests/input/
apache-log4j-1.2.15/tests/input/xml/
apache-log4j-1.2.15/tests/src/
apache-log4j-1.2.15/tests/src/java/
apache-log4j-1.2.15/tests/src/java/org/
.....................................
```

While untarring, it would create a directory hierarchy with a name apache-log4j-x.x.x as follows:

```
-rw-r--r--  1 root root   3565 2007-08-25 00:09 BUILD-INFO.txt
-rw-r--r--  1 root root   2607 2007-08-25 00:09 build.properties.sample
-rw-r--r--  1 root root  32619 2007-08-25 00:09 build.xml
drwxr-xr-x 14 root root   4096 2010-02-04 14:09 contribs
drwxr-xr-x  5 root root   4096 2010-02-04 14:09 examples
-rw-r--r--  1 root root   2752 2007-08-25 00:09 INSTALL
-rw-r--r--  1 root root   4787 2007-08-25 00:09 KEYS
-rw-r--r--  1 root root  11366 2007-08-25 00:09 LICENSE
-rw-r--r--  1 root root 391834 2007-08-25 00:29 log4j-1.2.15.jar
-rw-r--r--  1 root root    160 2007-08-25 00:09 NOTICE
```

```
-rwxr-xr-x  1 root root  10240 2007-08-25 00:27 NTEventLogAppender.dll
-rw-r--r--  1 root root  17780 2007-08-25 00:09 pom.xml
drwxr-xr-x  7 root root   4096 2007-08-25 00:13 site
drwxr-xr-x  8 root root   4096 2010-02-04 14:08 src
drwxr-xr-x  6 root root   4096 2010-02-04 14:09 tests
```

Step 2

This step is optional and depends on what features you are going to use from log4j framework. If you already have following packages installed on your machine then it is fine, otherwise you need to install them to make log4j work.

- **JavaMail API:** The e-mail based logging feature in log4j requires the Java Mail API (mail.jar) to be installed on your machine from **https://glassfish.dev.java.net/javaee5/mail/**.

- **JavaBeans Activation Framework:** The Java Mail API will also require that the JavaBeans Activation Framework (activation.jar) be installed on your machine from **http://java.sun.com/products/javabeans/jaf/index.jsp**.

- **Java Message Service:** The JMS-compatible features of log4j will require that both JMS and Java Naming and Directory Interface (JNDI) be installed on your machine from **http://java.sun.com/products/jms**.

- **XML Parser:** You need a JAXP-compatible XML parser to use log4j. Make sure you have Xerces.jar installed on your machine from **http://xerces.apache.org/xerces-j/install.html**.

Step 3

Now you need to set up the CLASSPATH and PATH variables appropriately. Here we are going to set it just for the log4j.x.x.x.jar file.

```
$ pwd
/usr/local/apache-log4j-1.2.15
$ export CLASSPATH= \
      $CLASSPATH:/usr/local/apache-log4j-1.2.15/log4j-1.2.15.jar
$ export PATH=$PATH:/usr/local/apache-log4j-1.2.15/
```

3. ARCHITECTURE

Log4j API follows a layered architecture where each layer provides different objects to perform different tasks. This layered architecture makes the design flexible and easy to extend in future.

There are two types of objects available with Log4j framework:

- **Core Objects:** These are mandatory objects of the framework. They are required to use the framework.
- **Support Objects:** These are optional objects of the framework. They support core objects to perform additional but important tasks.

Core Objects

Core objects include the following types of objects:

Logger Object

The top-level layer is the Logger which provides the Logger object. The Logger object is responsible for capturing logging information and they are stored in a namespace hierarchy.

Layout Object

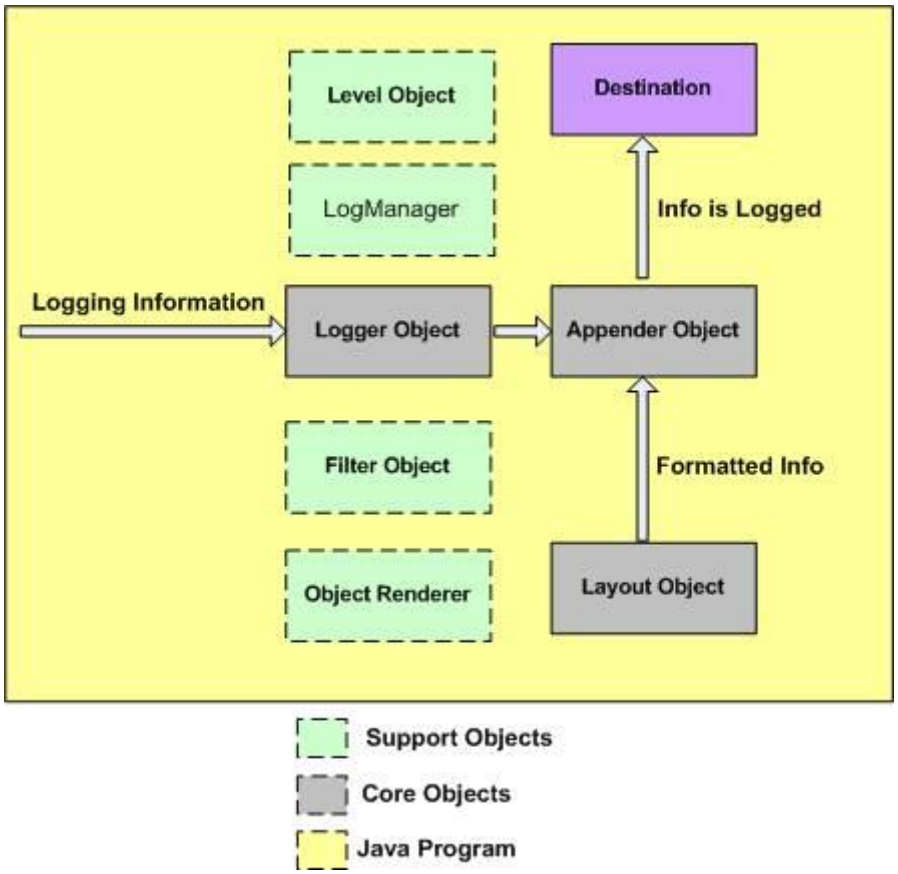
The Layout layer provides objects which are used to format logging information in different styles. It provides support to appender objects before publishing logging information.

Layout objects play an important role in publishing logging information in a way that is human-readable and reusable.

Appender Object

This is a lower-level layer which provides Appender objects. The Appender object is responsible for publishing logging information to various preferred destinations such as a database, file, console, UNIX Syslog, etc.

The following virtual diagram shows the components of a log4j framework:



Support Objects

There are other important objects in the log4j framework that play a vital role in the logging framework:

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.

4. CONFIGURATION

The previous chapter explained the core components of log4j. This chapter explains how you can configure the core components using a configuration file. Configuring log4j involves assigning the Level, defining Appender, and specifying Layout objects in a configuration file.

The *log4j.properties* file is a log4j configuration file which keeps properties in key-value pairs. By default, the LogManager looks for a file named *log4j.properties* in the CLASSPATH.

- The level of the root logger is defined as DEBUG. The DEBUG attaches the appender named X to it.
- Set the appender named X to be a valid appender.
- Set the layout for the appender X.

log4j.properties Syntax

Following is the syntax of *log4j.properties* file for an appender X:

```
# Define the root logger with appender X
log4j.rootLogger = DEBUG, X

# Set the appender named X to be a File appender
log4j.appender.X=org.apache.log4j.FileAppender

# Define the layout for X appender
log4j.appender.X.layout=org.apache.log4j.PatternLayout
log4j.appender.X.layout.conversionPattern=%m%n
```

log4j.properties Example

Using the above syntax, we define the following in *log4j.properties* file:

- The level of the root logger is defined as DEBUG. The DEBUG the appender named FILE to it.
- The appender FILE is defined as *org.apache.log4j.FileAppender*. It writes to a file named "log.out" located in the log directory.

- The layout pattern defined is *%m%n,* which means the printed logging message will be followed by a newline character.

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

It is important to note that log4j supports UNIX-style variable substitution such as ${variableName}.

Debug Level

We have used DEBUG with both the appenders. All the possible options are:

- TRACE
- DEBUG
- INFO
- WARN
- ERROR
- FATAL
- ALL

These levels would be explained in Log4j Logging Levels.

Appenders

Apache log4j provides Appender objects which are primarily responsible for printing logging messages to different destinations such as consoles, files, sockets, NT event logs, etc.

Each Appender object has different properties associated with it, and these properties indicate the behavior of that object.

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.

We can add an Appender object to a Logger by including the following setting in the configuration file with the following method:

```
log4j.logger.[logger-name]=level, appender1,appender..n
```

You can write same configuration in XML format as follows:

```
<logger name="com.apress.logging.log4j" additivity="false">
   <appender-ref ref="appender1"/>
   <appender-ref ref="appender2"/>
</logger>
```

If you are willing to add Appender object inside your program then you can use following method:

```
public void addAppender(Appender appender);
```

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 and JDBC Appender would be covered in Logging in Database.

Layout

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

- DateLayout
- HTMLLayout
- PatternLayout
- SimpleLayout
- XMLLayout

Using HTMLLayout and XMLLayout, you can generate log in HTML and in XML format as well.

Layout Formatting

You would learn how to format a log message in chapter: Log Formatting.

5. SAMPLE PROGRAM

We have seen how to create a configuration file. This chapter describes how to generate debug messages and log them in a simple text file.

Following is a simple configuration file created for our example. Let us revise it once again:

- The level of the root logger is defined as DEBUG and attaches appender named FILE to it.
- The appender FILE is defined as org.apache.log4j.FileAppender and writes to a file named "log.out" located in the **log** directory.
- The layout pattern defined is %m%n, which means the printed logging message will be followed by a newline character.

The contents of *log4j.properties* file are as follows:

```
# Define the root logger with appender file
log = /usr/home/log4j
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

Using log4j in Java Program

The following Java class is a very simple example that initializes and then uses the Log4J logging library for Java applications.

```
import org.apache.log4j.Logger;

import java.io.*;
```

```
import java.sql.SQLException;
import java.util.*;

public class log4jExample{
  /* Get actual class name to be printed on */
  static Logger log = Logger.getLogger(log4jExample.class.getName());

  public static void main(String[] args)
                throws IOException,SQLException{

     log.debug("Hello this is  a debug message");
     log.info("Hello this is an info message");
  }
}
```

Compile and Execute

Here are the steps to compile and run the above-mentioned program. Make sure you have set PATH and CLASSPATH appropriately before proceeding for the compilation and execution.

All the libraries should be available in CLASSPATH and your *log4j.properties* file should be available in PATH. Follow the steps given below:

- Create log4j.properties as shown above.
- Create log4jExample.java as shown above and compile it.
- Execute log4jExample binary to run the program.

You would get the following result inside /usr/home/log4j/log.out file:

```
Hello this is a debug message
Hello this is an info message
```

6. LOGGING METHODS

Logger class provides a variety of methods to handle logging activities. The Logger class does not allow us to instantiate a new Logger instance but it provides two static methods for obtaining a Logger object:

- public static Logger getRootLogger();
- public static Logger getLogger(String name);

The first of the two methods returns the application instance's root logger and it does not have a name.

Any other named Logger object instance is obtained through the second method by passing the name of the logger. The name of the logger can be any string you can pass, usually a class or a package name as we have used in the last chapter and it is mentioned below:

```
static Logger log = Logger.getLogger(log4jExample.class.getName());
```

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.

Sr. No.	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.

All the levels are defined in the org.apache.log4j.Level class and any of the above-mentioned methods can be called as follows:

```
import org.apache.log4j.Logger;

public class LogClass {
   private static org.apache.log4j.Logger log = Logger
                                    .getLogger(LogClass.class);
   public static void main(String[] args) {
      log.trace("Trace Message!");
      log.debug("Debug Message!");
      log.info("Info Message!");
      log.warn("Warn Message!");
      log.error("Error Message!");
      log.fatal("Fatal Message!");
   }
}
```

When you compile and run LogClass program, it would generate the following result:

```
Debug Message!
Info Message!
Warn Message!
Error Message!
```

```
Fatal Message!
```

All the debug messages make more sense when they are used in combination with levels. We will cover levels in the next chapter and then, you would have a good understanding of how to use these methods in combination with different levels of debugging.

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**