Log4j教程

极客学院出版

前言

Log4j 是一个使用 Java 语言编写的，可靠、快速、灵活的日志框架（API），使用 Apache Software License 授权。Log4j 是一个使用 Java 语言编写的流行类库，它被移植到 C、C++、C#、Perl、Python、Ruby 和 Eiffel 语言中。

适用人群

该教程适用于初学者，帮助他们掌握 Log4j 的基本功能。

学习前提

在很多使用 Java 编写的应用中，都会用到 Log4j，因此，读者对 Java 编程语言要有良好的掌握。

更新日期	更新内容
2015-05-19	Log4j 中文版

版本信息

书中演示代码基于以下版本：

语言/框架	版本信息
apache-log4j	1.2.15

目录

Log4j 教程

概述

Log4j 是一个使用 Java 语言编写的，可靠、快速、灵活的日志框架（API），使用 Apache Software License 授权。

它被移植到 C、C++、C#、Perl、Python、Ruby 和 Eiffel 语言中。

Log4j 是高度可配置的，在运行期使用外部的配置文件对其进行配置。它按照优先级别记录日志，并可将日志信息定向输出到各种介质，比如数据库、文件、控制台、Unix Syslog等。

Log4j 主要由三部分组成：

- loggers：负责采集日志信息。
- appenders：负责将日志信息发布到不同地方。
- layouts：负责以各种风格格式化日志信息。

Log4j 的历史

- 始于 1996 年，作为记录E.U. SEMPER (Secure Electronic Marketplace for Europe)项目跟踪信息的 API。
- 经过大量的完善和蜕变，最初的 API 终于演进为 Log4j，一个在 Java 社区流行的日志类库。
- 该类库使用 Apache Software License 授权，该授权是经开源促进协会认证的、完整的开源协议。
- 最新版本的 Log4j，连同其代码、类文件和文档可通过 http://logging.apache.org/log4j/ 获取。

Log4j 的功能

- 线程安全。
- 速度优化。
- 基于命名的 logger 层次。
- 每个 logger 支持多种输出 appender。
- 支持国际化。
- 不受限于预定义好的设施。

- 日志记录行为可在运行期通过配置文件设置。
- 设计之初就考虑了处理 Java 异常。
- 使用多个日志级别：ALL、TRACE、DEBUG、INFO、WARN、ERROR、FATAL。
- 通过扩展 `Layout` 类可轻松改变输出日志的格式。
- 输出日志的目的和策略可通过实现 `Appender` 接口改变。
- 失败即停止。虽然 Log4j 努力做到最好，但不保证每一条日志都能发送到指定目的地。

记录日志的优缺点

记录日志是软件开发中的重要一环。编写良好的日志代码能为运行应用提供快速的诊断信息和良好的存储结构，方便维护。

记录日志也有其缺点，它会让应用变慢。假如输出太详细，可能会导致屏幕闪动。为了减轻这些影响，Log4j 被设计为可靠的，更快的和可扩展的。

由于日志很少是应用程序关注的焦点，所以 Log4j API 力争做到简单并易于理解和使用。

安装

Log4j API 使用 Apache Software License 授权，该授权是经开源促进协会认证的、完整的开源协议。

最新版本的 Log4j，连同其代码、类文件和文档可通过 http://logging.apache.org/log4j/ 获取。

从上述地址下载 apache-Log4j-x.x.x.tar.gz 文件，按照下面的步骤安装 Log4j。

第一步

将所下载文件解压至 `/usr/local/` 目录：

```
$ 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/
.....................................
```

解压时，会生成一个名为 `apache-log4j-x.x.x` 的文件夹，目录结构如下所示：

```
-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
```

第二步

该步根据要使用 log4j 的功能是可选的。如果您已经在机器上安装了如下软件包，就不用担心了，否则您需要安装它们以使 Log4j 工作正常。

- JavaMail API：Log4j 中基于 e-mail 的功能需要 Java Mail API（mail.jar），请从 glassfish.dev 下载安装。
- JavaBeans Activation Framework：Java Mail API 还需要安装 JavaBeans Activation Framework（activation.jar），请从 http://java.sun.com/products/javabeans/jaf/index.jsp 下载安装。
- Java Message Service：Log4j 中和 JMS 匹配的功能需要安装 JMS 和 Java Naming and Directory Interface JNDI，请从 http://java.sun.com/products/jms 下载安装。
- XML Parser：使用 Log4j，需要一个和JAXP兼容的 XML 解析器，请确保从 http://xerces.apache.org/xerces-j/install.html 下载安装了 Xerces.jar。

第三步

现在需要正确地设置 CLASSPATH 和 PATH。这里我们只如何设置 Log4j.x.x.x.jar。

```
$ 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/
```

架构

Log4j API 采用分层架构，每一层有不同的对象，完成不同的任务。这种分层架构让设计变得灵活，且易于日后扩展。

Log4j 里有两种类型的对象：

- **核心对象**：这是框架必需的对象，使用框架时必需用到它们。
- **支持对象**：这是框架的可选对象，它们支持核心对象做一些额外的，但并不重要的任务。

核心对象

核心对象包括如下几种类型：

Logger 对象

最上一层是 Logger 对象，该对象负责获取日志信息，并存储于一个分层的命名空间之中。

Layout 对象

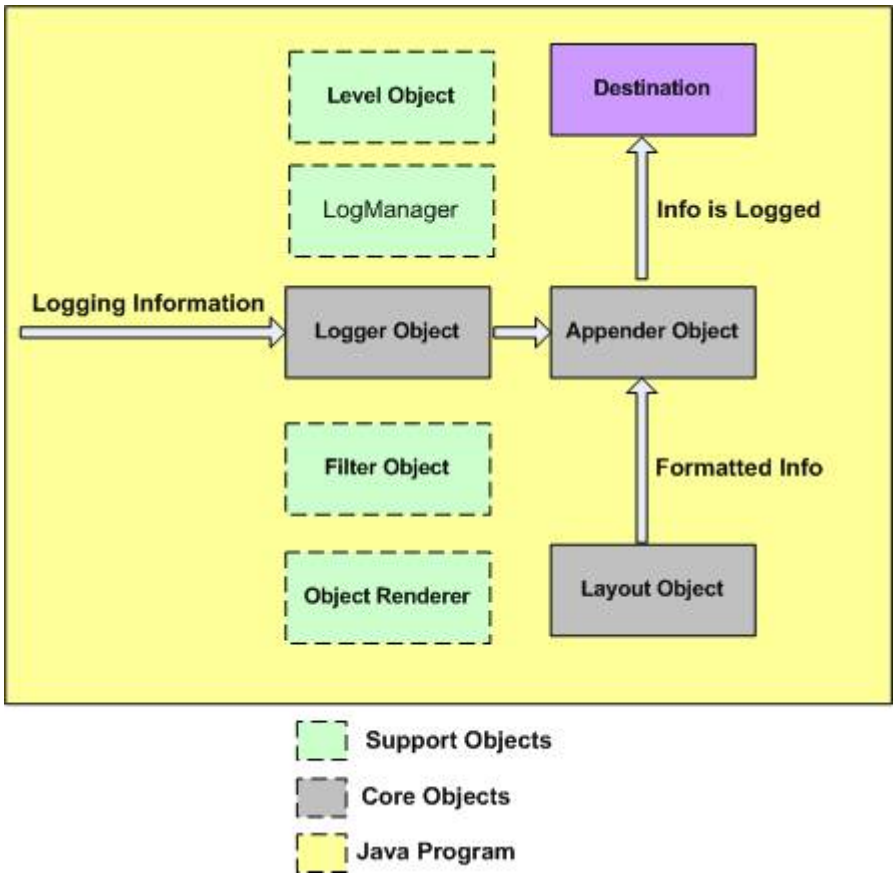
Layout 层提供了用于以各种风格格式化日志信息的对象，在发布日志信息之前，它为 appender 对象提供支持。

Layout 对象对于发布日志信息非常重要，它使日志变得可读、可复用。

Appender 对象

该对象位于分层架构中的较低一层，Appender 对象负责将日志信息发布到不同目的地，比如数据库、文件、控制台、Unix Syslog 等。

下图展示了 Log4j 框架中的组件：



支持对象

Log4j 中还有其他一些对象，同样在框架中发挥着重要作用。

Level 对象

Level 对象定义了日志信息的粒度和优先级。API 定义了七种级别：OFF、DEBUG、INFO、ERROR、WARN、FATAL、ALL。

Filter 对象

Filter 对象用来分析日志信息，进而决定该条日志是否被记录。

一个 Appender 对象可对应多个 Filter 对象，当日志信息传给 Appender 对象时，与其关联的所有 Filter 对象需要判断是否将日志信息发布到目的地。

ObjectRenderer

ObjectRenderer 对象负责为传入日志框架的不同对象提供字符串形式的表示，Layout 对象使用该对象来准备最终的日志信息。

LogManager

LogManager 对象管理日志框架，它负责从系统级的配置文件或类中读取初始配置参数。

配置

上一章解释了 Log4j 的核心组件。本章讲述如何使用配置文件来配置核心组件。Log4j 的配置包括在配置文件中指定 Level、定义 Appender 和指明 Layout。

Log4j.properties 文件是 Log4j 的配置文件，属性以键值对的形式定义。默认情况下，LogManager 会在 CLASSPATH 中寻找 Log4j.properties 文件。

- 根日志的级别定义为 DEBUG，并将名为 X 的 appender 添加其上。
- 将名为 X 的 appender 设置为合法的 appender。
- 设置 appender X 的 layout。

Log4j.properties 的语法

为 appender X 定义的 *Log4j.properties* 的语法如下：

```
 # 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 示例

使用上述语法，我们在 Log4j.properties 定义如下属性：

- 定义根日志级别为 DEBUG，并将名为 FILE 的 appender 添加其上。
- appender FILE 定义为 org.apache.Log4j.FileAppender，它将日志写入 log 目录下一个名为 log.out 的文件中。
- layout 被定义为 *%m%n*，即打印出来的日志信息末尾加入换行。

```
 # 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
```

值得注意的是，Log4j 支持 Unix 风格的变量替换，比如 ${variableName}。

DEBUG 级别

两个 appender 都使用了 DEBUG 级别，所有可能的选项如下：

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

这些级别会在本教程的后续部分讲解。

Appender

Apache Log4j 提供的 Appender 对象主要负责将日志信息打印至不同目的地，比如控制台、文件、网络套接字、NT 事件日志等。

每个 Appender 对象都有不同的属性，这些属性决定了该对象的行为。

属性	描述
layout	Appender 使用 Layout 对象和与之关联的模式来格式化日志信息。

target	目的地可以是控制台、文件，或依赖于 appender 的对象。
level	级别用来控制过滤日志信息。
threshold	Appender 可脱离于日志级别定义一个阀值级别，Appender 对象会忽略所有级别低于阀值级别的日志。
filter	Filter 对象可在级别基础之上分析日志信息，来决定 Appender 对象是否处理或忽略一条日志记录。

通过在配置文件中使用如下方法，将 Appender 对象添加至 Logger 对象：

```
Log4j.logger.[logger-name]=level, appender1,appender..n
```

也可以在 XML 中定义同样的配置：

```
<logger name="com.apress.logging.Log4j" additivity="false">
  <appender-ref ref="appender1"/>
  <appender-ref ref="appender2"/>
</logger>
```

如果想在程序中添加 Appender 对象，可使用如下方法：

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

`addAppender()` 方法为 Logger 对象增加一个 Appender。如示例配置展示的那样，可以通过逗号分隔的列表，为 logger 添加多个 Appender，将日志信息打印到不同的目的地。

在上面的例子中，我们只用到了 *FileAppender*，所有可用的 appender 包括：

- AppenderSkeleton
- AsyncAppender
- ConsoleAppender
- DailyRollingFileAppender
- ExternallyRolledFileAppender
- FileAppender
- JDBCAppender
- JMSAppender
- LF5Appender
- NTEventLogAppender
- NullAppender
- RollingFileAppender

- SMTPAppender
- SocketAppender
- SocketHubAppender
- SyslogAppender
- TelnetAppender
- WriterAppender

我们将在使用文件记录日志 (页 0)一章讲解 JDBC Appender。

Layout

我们已经在 appender 中使用了 PatternLayout，所有选项还包括：

- DateLayout
- HTMLLayout
- PatternLayout
- SimpleLayout
- XMLLayout

使用 HTMLLayout 和 XMLLayout，可以生成 HTML 和 XML 格式的日志。

日志格式化

您将在日志格式一章学习如何格式化日志信息。

示例程序

我们已经学会了如何创建配置文件，本章讲述如何生成调试信息，并将其写入一个简单的文本文件。

下面是为我们的例子创建的一个简单配置文件，让我们再来复习一遍：

- 定义根日志级别为 DEBUG，并将名为 FILE 的 appender 添加其上。
- appender FILE 定义为 org.apache.Log4j.FileAppender，它将日志写入 log 目录下一个名为 log.out 的文件中。
- layout 被定义为 *%m%n*，即打印出来的日志信息末尾加入换行。

Log4j.properties 文件的内容如下：

```
 # 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
```

在 Java 程序中使用 Log4j

下面的 Java 类是一个非常简单的例子，它在 Java 应用中初始化，然后使用了 Log4j 类库。

```
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");
  }
}
```

编译和运行

下面是编译和运行上述程序的步骤。在编译和运行前，首先确保正确地设置了 CLASSPATH 和 PATH。

所有的类库都必需包含在 CLASSPATH 里，Log4j.properties 文件必需包含在 PATH 里。按照下面的步骤操作：

- 创建如上所述的 Log4j.properties 文件。
- 创建如上所述的 Log4jExample.java 文件并编译。
- 运行 Log4jExample。

在 `/usr/home/Log4j/log.out` 文件中会生成如下内容：

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

Logging 方法

Logger 类提供了很多方法用来处理日志，Logger 类不允许初始化一个新的实例，但提供了两个静态方法用来获取 Logger 对象：

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

第一个方法返回应用实例没有名字的根日志。

其他有名字的 Logger 对象通过传入日志的名字，调用第二个方法获得。日志的名字是传入的任何字符串，通常为类名或包名，如上一章和下面的例子所示：

```
static Logger log = Logger.getLogger(Log4jExample.class.getName());
```

Logging 方法

一旦获取一个有名字的 logger 实例，就可以使用多个方法记录日志。Logger 类拥有如下方法用于打印日志信息。

#	方法和描述
1	public void debug(Object message) 使用 Level.DEBUG 级别打印日志。
2	public void error(Object message) 使用 Level.ERROR 级别打印日志。
3	public void fatal(Object message) 使用 Level.FATAL 级别打印日志。
4	public void info(Object message) 使用 Level.INFO 级别打印日志。
5	public void warn(Object message) 使用 Level.WARN 级别打印日志。

6	public void trace(Object message) 使用 Level.TRACE 级别打印日志。

所有级别均在 org.apache.Log4j.Level 类中定义，这些方法使用如下方式调用：

```
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!");
  }
}
```

编译并运行 LogClass，输出如下：

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

调试信息和级别联合使用才更有意义。我们将在下一章讲解日志级别，您会对如何联合使用这些方法和不同调试级别有一个更好的理解。

Logging 级别

org.apache.Log4j.Level 类定义了日志级别，您可通过继承 Level 类定制自己的级别。

级别	描述
ALL	所有级别，包括定制级别。
DEBUG	指明细致的事件信息，对调试应用最有用。
ERROR	指明错误事件，但应用可能还能继续运行。
FATAL	指明非常严重的错误事件，可能会导致应用终止执行。
INFO	指明描述信息，从粗粒度上描述了应用运行过程。
OFF	最高级别，用于关闭日志。
TRACE	比 DEBUG 级别的粒度更细。
WARN	指明潜在的有害状况。

级别是如何工作的?

在一个级别为 q 的 logger 对象中，一个级别为 p 的日志请求在 p >= q 的情况下是**开启**的。该规则是 Log4j 的核心，它假设级别是有序的。对于标准级别，其顺序为：ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF。

下面的例子展示了如何过滤 DEBUG 和 INFO 级别的日志。改程序使用 logger 对象的 setLevel(Level.X) 方法设置期望的日志级别：

该例子会打印出除过 DEBUG 和 INFO 级别外的所有信息：

```
import org.apache.Log4j.*;

public class LogClass {
  private static org.apache.Log4j.Logger log = Logger.getLogger(LogClass.class);

  public static void main(String[] args) {
    log.setLevel(Level.WARN);

    log.trace("Trace Message!");
    log.debug("Debug Message!");
    log.info("Info Message!");
    log.warn("Warn Message!");
    log.error("Error Message!");
    log.fatal("Fatal Message!");
```

```
  }
}
```

编译并运行 `LogClass`，会产生如下输出：

```
Warn Message!
Error Message!
Fatal Message!
```

使用配置文件设置日志级别

Log4j 提供了基于配置文件设置日志级别的功能，当您需要改变调试级别时，不用再去修改代码了。

下面这个例子和上面那个例子功能一样，不过不用使用 `setLevel(Level.WARN)` 方法，只需修改配置文件：

```
 # Define the root logger with appender file

log = /usr/home/Log4j
Log4j.rootLogger = WARN, 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
```

现在使用如下程序：

```
import org.apache.Log4j.*;

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!");
   }
}
```

编译并运行如下程序，会在 /usr/home/Log4j/log.out 文件内生成如下内容：

```
Warn Message!
Error Message!
Fatal Message!
```

日志格式

Apache Log4j 提供了多个 Layout 对象，每个根据布局的不同都可格式化日志数据。还可以创建一个 Layout 对象，以应用特有的方式格式化日志。

所有 Layout 对象从 Appender 对象那里接收一个 LoggingEvent 对象，然后从 LoggingEvent 对象那里获取信息，并使用恰当的 ObjectRenderer 对象获取该信息的字符串形式。

Layout 类型

位于继承关系顶层的是抽象类 org.apache.Log4j.Layout，这是所有 Log4j API 中 Layout 类的基类。

Layout 类是个抽象类，在应用中我们从不直接使用该类，而是使用它的子类，如下所示：

- DateLayout
- HTMLLayout
- PatternLayout
- SimpleLayout
- XMLLayout

Layout 方法

该类对于所有其他 Layout 对象的通用操作提供了框架性的实现，声明了两个抽象方法：

序号	方法 & 描述
1	public abstract boolean ignoresThrowable() 该方法指明日志信息是否处理由日志事件传来的 java.lang.Throwable 对象，如果 Layout 对象能处理 Throwable 对象，则 Layout 对象不会忽略它，并且返回 false。
2	public abstract String format(LoggingEvent event) 各 Layout 子类实现该方法，以定义各自的格式。

除了这些抽象方法，`Layout` 类还提供了如下的具体方法：

序号	方法 & 描述
1	public String getContentType() 返回 Layout 对象使用的内容类型。基类返回 text/plain 作为默认内容类型。
2	public String getFooter() 返回日志信息的脚注。
3	public String getHeader() 返回日志信息的日志头。

每个子类均可覆盖这些方法的实现，返回各自特有的信息。

使用文件记录日志

使用 `org.apache.Log4j.FileAppender` 将日志记录到文件。

FileAppender 配置

FileAppender 拥有如下配置参数：

属性	描述
immediateFlush	该标志位默认为 true，意味着每次日志追加操作都将输出流刷新至文件。
encoding	可以使用任何编码，默认情况下使用平台相关的编码。
threshold	appender 对象的阀值。
Filename	日志文件名。
fileAppend	该值默认为 true，其含义是让日志追加至文件末尾。
bufferedIO	该标志位表示是否打开缓冲区写，缺省为 false。
bufferSize	如果开启缓冲区 I/O，该属性指示缓冲区大小，缺省为 8 kb。

下面是一个使用 FileAppender 的示例配置文件 Log4j.properties：

```
 # 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
```

如果您需要一个和 Log4j.properties 文件功能类似的 XML 配置文件，如下所示：

```
<?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>
```

您可以使用上述配置尝试示例程序一章中的例子。

使用多个文件记录日志

您可能因为某些原因，需要将日志写入多个文件，比如当文件大小达到一定阀值时。

为了将日志写入多个文件，您需要使用 `org.apache.Log4j.RollingFileAppender`，该类继承了 `FileAppender` 类，继承了它的所有属性。

除了上述提到的 `FileAppender` 类的属性，该类还包括如下可配置的属性：

属性	描述
maxFileSize	这是文件大小的关键值，大于该值时，文件会回滚。该值默认为 10 MB。
maxBackupIndex	该值表示备份文件的个数，默认为 1。

下面是为 RollingFileAppender 定义的示例配置文件 Log4j.properties：

```
# 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
```

如果您想使用 XML 配置文件，可以像上一节一样生成 XML 配置文件，只需添加和 RollingFileAppender 相关的配置即可。

该示例配置文件展示了每个日志文件最大为 5 MB，如果超过该最大值，则会生成一个新的日志文件。由于 `maxBackupIndex` 的值为 2，当第二个文件的大小超过最大值时，会擦去第一个日志文件的内容，所有的日志都回滚至第一个日志文件。

您可以使用上述配置尝试示例程序一章中的例子。

逐日生成日志文件

您也许需要逐日生成日志文件，以此更加整洁的记录日志信息。

为了逐日生成日志文件，需要使用 `org.apache.Log4j.DailyRollingFileAppender` ，，该类继承了 `FileAppender` 类，继承了它的所有属性。

除了上述提到的 `FileAppender` 类的属性，该类多包含了一个重要属性：

属性	描述
DatePattern	该属性表明什么时间回滚文件，以及文件的命名约定。缺省情况下，在每天午夜回滚文件。

`DatePattern` 使用如下规则控制回滚计划：

DatePattern	描述
'.' yyyy-MM	在本月末，下月初回滚文件。
'.' yyyy-MM-dd	在每天午夜回滚文件，这是缺省值。
'.' yyyy-MM-dd-a	在每天中午和午夜回滚文件。
'.' yyyy-MM-dd-HH	在每个整点回滚文件。
'.' yyyy-MM-dd-HH-mm	每分钟回滚文件。
'.' yyyy-ww	根据地域，在每周的第一天回滚。

下述 Log4j.properties 示例文件产生的日志文件在每天中午和午夜回滚：

```
 # 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
```

如果您想使用 XML 配置文件，可以像上一节一样生成 XML 配置文件，只需添加和 DailyRollingFileAppender 相关的配置即可。

您可以使用上述配置尝试示例程序一章中的例子。

使用数据库记录日志

Log4j API 提供了 org.apache.Log4j.jdbc.JDBCAppender 对象，该对象可将日志信息记录到特定的数据库之中。

JDBCAppender 配置

属性	描述
bufferSize	设置缓冲区大小，缺省为 1。
driver	以字符串形式设置驱动类，如果未设置，缺省为 sun.jdbc.odbc.JdbcOdbcDriver。
layout	设置 layout，缺省为 org.apache.Log4j.PatternLayout。
password	设置数据库密码。
sql	设置每次日志事件触发时需要执行的 SQL 语句，该语句可以是 INSERT、UPDATE 或 DELETE。
URL	设置 JDBC URL.
user	设置数据库用户名。

日志表的配置

在使用基于 JDBC 的日志之前，先要创建一张表以保存所有日志信息，下面是用来创建 LOGS 表的 SQL 语句：

```
CREATE TABLE LOGS
  (USER_ID VARCHAR(20)   NOT NULL,
   DATED   DATE          NOT NULL,
   LOGGER  VARCHAR(50)   NOT NULL,
   LEVEL   VARCHAR(10)   NOT NULL,
   MESSAGE VARCHAR(1000) NOT NULL
  );
```

示例配置文件

下面是一个为 JDBCAppender 编写的 Log4j.properties 的示例配置文件，使用该对象将日志信息记录到 LOGS 表中。

```
 # Define the root logger with appender file

Log4j.rootLogger = DEBUG, DB

 # Define the DB appender

Log4j.appender.DB=org.apache.Log4j.jdbc.JDBCAppender

 # Set JDBC URL

Log4j.appender.DB.URL=jdbc:mysql://localhost/DBNAME

 # Set Database Driver

Log4j.appender.DB.driver=com.mysql.jdbc.Driver

 # Set database user name and password

Log4j.appender.DB.user=user_name
Log4j.appender.DB.password=password

 # Set the SQL statement to be executed.

Log4j.appender.DB.sql=INSERT INTO LOGS VALUES('%x','%d','%C','%p','%m')

 # Define the layout for file appender

Log4j.appender.DB.layout=org.apache.Log4j.PatternLayout
```

如果使用 MySQL 数据库，需要使用真实的 DBNAME、用户名和密码，就是刚才用来创建 LOGS 表的那些属性。SQL 语句执行 INSERT 语句，为 LOGS 表插入具体数值。

`JDBCAppender` 不需要显示定义 layout，传入的 SQL 语句会使用 `PatternLayout` 。

如果您需要和上述 Log4j.properties 文件等价的 XML 配置文件，如下所示：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Log4j:configuration SYSTEM "Log4j.dtd">
<Log4j:configuration>

<appender name="DB" class="org.apache.Log4j.jdbc.JDBCAppender">
  <param name="url" value="jdbc:mysql://localhost/DBNAME"/>
  <param name="driver" value="com.mysql.jdbc.Driver"/>
  <param name="user" value="user_id"/>
  <param name="password" value="password"/>
  <param name="sql" value="INSERT INTO LOGS VALUES('%x','%d','%C','%p','%m')"/>
```

```
   <layout class="org.apache.Log4j.PatternLayout">
   </layout>
</appender>

<logger name="Log4j.rootLogger" additivity="false">
   <level value="DEBUG"/>
   <appender-ref ref="DB"/>
</logger>

</Log4j:configuration>
```

示例程序

下述 Java 类是一个非常简单的例子，该类在 Java 应用中初始化并使用了 Log4j 类库。

```
import org.apache.Log4j.Logger;
import java.sql.*;
import java.io.*;
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("Debug");
      log.info("Info");
   }
}
```

编译和运行上述程序的步骤如下。在继续编译和运行程序之前，确保正确设置了 PATH 和 CLASSPATH。

所有的类库都需要包含在 CLASSPATH 中，*Log4j.properties* 文件需要包含在 PATH 中，步骤如下：

- 创建如上所示的 Log4j.properties 文件。
- 创建如上所示的 Log4jExample.java 文件并编译。
- 运行 Log4jExample。

现在检查 DBNAME 数据库中的 LOGS 表，会发现如下条目：

```
mysql >  select * from LOGS;
+---------+------------+--------------+-------+---------+
```

```
| USER_ID | DATED     | LOGGER      | LEVEL | MESSAGE |
+---------+-----------+-------------+-------+---------+
|         | 2010-05-13 | Log4jExample | DEBUG | Debug   |
|         | 2010-05-13 | Log4jExample | INFO  | Info    |
+---------+-----------+-------------+-------+---------+
2 rows in set (0.00 sec)
```

注意——这里 x 用来输出和生成日志事件线程相关联的嵌套诊断上下文（NDC），我们使用 NDC 在处理多个客户端的服务器端来区分客户端，具体请查阅 Log4j 手册。

2 附录

HTMLLayout

如果您希望以 HTML 格式输出日志文件，可使用 org.apache.Log4j.HTMLLayout 格式化日志信息。

HTMLLayout 继承自抽象类 org.apache.Log4j.Layout，覆盖了其 format() 方法来提供 HTML 格式。

它提供了如下信息以供显示：

- 从应用启动到日志产生之间经过的时间。
- 发起记录日志请求的线程名字。
- 和记录日志请求关联的级别。
- logger 的名字和日志信息。
- 程序文件的位置信息和触发日志的行号，该信息是可选的。

HTMLLayout 是一种非常简单的 Layout 对象，它提供了如下方法：

序号	方法 & 描述
1	setContentType(String) 设置 HTML 的内容类型，默认为 text/html。
2	setLocationInfo(String) 设置日志事件的地域信息。
3	setTitle(String) 设置 HTML 文件的标题，默认为 Log4j Log Messages。

HTMLLayout 示例

下面是为 HTMLLayout 编写的一个简单配置文件：

```
 # 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}/htmlLayout.html

 # Define the layout for file appender

Log4j.appender.FILE.layout=org.apache.Log4j.HTMLLayout
Log4j.appender.FILE.layout.Title=HTML Layout Example
Log4j.appender.FILE.layout.LocationInfo=true
```

下面的 Java 程序会生成日志信息：

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

编译和运行上述程序，在目录 /usr/home/Log4j 下，会生成一个名为 htmlLayout.html 的文件，该文件包含如下日志信息：

Log session start time Mon Mar 22 13:30:24 AST 2010

Time	Thread	Level	Category	File:Line	Message
0	main	DEBUG	Log4jExample	Log4jExample.java:15	Hello this is an debug message
6	main	INFO	Log4jExample	Log4jExample.java:16	Hello this is an info message

您可以使用浏览器打开 htmlLayout.html 文件。需要注意的是末尾缺失了 `</html>` 和 `</body>` 标签。

将日志格式化为 HTML 的一个优势在于可将其发布成一个 Web 页面，便于远程浏览。

PatternLayout

如果您希望基于某种模式生成特定格式的日志信息，可使用 org.apache.Log4j.PatternLayout 格式化您的日志信息。

PatternLayout 继承自抽象类 org.apache.Log4j.Layout，覆盖了其 format() 方法，通过提供的模式，来格式化日志信息。

PatternLayout 是一个简单的 Layout 对象，提供了如下属性，该属性可通过配置文件更改：

序号	属性 & 描述
1	conversionPattern 设置转换模式，默认为 %r [%t] %p %c %x - %m%n。

模式转换字符

下面的表格解释了上面模式中用到的字符，以及所有定制模式时能用到的字符：

转换字符	含义
c	使用它为输出的日志事件分类，比如对于分类 "a.b.c"，模式 %c{2} 会输出 "b.c" 。
C	使用它输出发起记录日志请求的类的全名。比如对于类 "org.apache.xyz.SomeClass"，模式 %C{1} 会输出 "SomeClass"。
d	使用它输出记录日志的日期，比如 %d{HH:mm:ss,SSS} 或 %d{dd MMM yyyy HH:mm:ss,SSS}。
F	在记录日志时，使用它输出文件名。
l	用它输出生成日志的调用者的地域信息。
L	使用它输出发起日志请求的行号。
m	使用它输出和日志事件关联的，由应用提供的信息。
M	使用它输出发起日志请求的方法名。
n	输出平台相关的换行符。
p	输出日志事件的优先级。
r	使用它输出从构建布局到生成日志事件所花费的时间，以毫秒为单位。
t	输出生成日志事件的线程名。
x	输出和生成日志事件线程相关的 NDC (嵌套诊断上下文)。
X	该字符后跟 MDC 键，比如 X{clientIP} 会输出保存在 MDC 中键 clientIP 对应的值。
%	百分号， %% 会输出一个 %。

格式修饰符

缺省情况下，信息保持原样输出。但是借助格式修饰符的帮助，就可调整最小列宽、最大列宽以及对齐。

下面的表格涵盖了各种修饰符：

格式修饰符	左对齐	最小宽度	最大宽度	注释
%20c	否	20	无	如果列名少于 20 个字符，左边使用空格补齐。
%-20c	是	20	无	如果列名少于 20 个字符，右边使用空格补齐。
%.30c	不适用	无	30	如果列名长于 30 个字符，从开头剪除。
%20.30c	否	20	30	如果列名少于 20 个字符，左边使用空格补齐，如果列名长于 30 个字符，从开头剪除。
%-20.30c	是	20	30	如果列名少于 20 个字符，右边使用空格补齐，如果列名长于 30 个字符，从开头剪除。

PatternLayout 示例

下面是为 `PatternLayout` 编写的一个简单配置：

```
 # 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=%d{yyyy-MM-dd}-%t-%x-%-5p-%-10c:%m%n
```

下面是生成日志信息的 Java 程序：

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

编译并运行上述程序，会在目录 `/usr/home/Log4j` 下生成一个名为 `log.out` 的文件，该文件包含如下日志信息：

```
2010-03-23-main--DEBUG-Log4jExample:Hello this is an debug message
2010-03-23-main--INFO -Log4jExample:Hello this is an info message
```