**log4j**

# log4j概述

log4j是一个用Java编写的可靠，快速和灵活的日志框架（API），它在Apache软件许可下发布。

log4j已经被移植到了C，C++，C＃，Perl，Python和Ruby等语言中。

log4j是高度可配置的，并可通过在运行时的外部文件配置。它根据记录的优先级别，并提供机制，以指示记录信息到许多的目的地，诸如：数据库，文件，控制台，UNIX系统日志等。

log4j中有三个主要组成部分：

1. loggers: 负责捕获记录信息。
2. appenders : 负责发布日志信息，以不同的首选目的地。
3. layouts: 负责格式化不同风格的日志信息。

**log4j的历史**

1. 始于1996年初的E.U. SEMPER （安全电子市场为欧洲）跟踪API的项目。
2. 不计其数的改进，在几个大量的工作之后，API已经发展成为log4j，一个流行的日志记录包为Java。
3. 该软件包是Apache软件许可证，由开源认证是一个不折不扣的开源许可证下发布。
4. 最新log4j的版本，包括完整的源代码，类文件和文档可以在这里找到 http://logging.apache.org/log4j/.

**log4j 特性:**

1. log4j的是线程安全的
2. log4j是经过优化速度的
3. log4j是基于一个名为记录器的层次结构
4. log4j的支持每个记录器多输出追加器（appender）
5. log4j支持国际化。
6. log4j并不限于一组预定义的设备
7. 日志行为可以使用配置文件在运行时设置
8. log4j设计从一开始就是处理Java异常
9. log4j使用多个层次，即ALL，TRACE，DEBUG，INFO，WARN，ERROR和FATAL
10. 日志输出的格式可以通过扩展Layout类容易地改变
11. 日志输出的目标，以及在写入策略可通过实现Appender程序接口改变
12. log4j 会故障停止。然而，尽管它肯定努力确保传递，log4j不保证每个日志语句将被传递到目的地。

**日志记录N个缺点及优点：**

日志是软件开发的重要组成部分。一个精心编写的日志代码提供快速的调试，维护方便，以及应用程序的运行时信息结构化存储。

日志记录确实也有它的缺点。它可以减缓的应用程序。如果太详细，它可能会导致滚动失明。为了减轻这些影响，log4j被设计为是可靠，快速和可扩展。

由于记录很少为应用的主要重点，但log4j API致力于成为易于理解和使用。

# log4j安装配置

log4j的API包使用Apache软件许可证，由开源倡议认证一个完全成熟的开源许可证下发布。

最新log4j的版本，包括完整的源代码，类文件和文档可以在这里找到 http://logging.apache.org/log4j/.

下载 apache-log4j-x.x.x.tar.gz 做到以下几点：

**步骤1:**

将下载的文件解压缩和解压在 /usr/local/ 目录，如下所示：

$ 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/

.......................................

当执行解压缩，这将创建一个名称 apache-log4j-x.x.x 的目录层次结构如下：

-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

**步骤2:**

这一步是可选的，取决于什么功能，要使用log4j 框架。如果已经有以下安装在您的机器，那么可以使用这些软件包，否则将需要安装它们，才能正常使log4j工作

1. JavaMail API: 电子邮件。基于log4j日志记录功能需要Java邮件API（mail.jar）在计算机上安装 https://glassfish.dev.java.net/javaee5/mail/
2. JavaBeans 活动框架： Java邮件API还需要JavaBeans激活框架（的activation.jar）从http://java.sun.com/products/javabeans/jaf/index.jsp 上下载安装在您的计算机上
3. Java 信息服务: log4j的JMS兼容的功能将需要两个JMS和JNDI（Java命名和Directory接口），可以从http://java.sun.com/products/jms 下载安装
4. XML Parser: 需要一个JAXP兼容的XML解析器来使用log4j。请确保已经在机器上安装xerces.jar ，可从http://xerces.apache.org/xerces-j/install.html 下载安装

**步骤3:**

这一步是非常重要的，需要设置CLASSPATH和PATH变量正确。在这里要设置只是 log4j.x.x.x.jar 文件

$ 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/

注意：如果在Window上开发使用Eclipse的话，可以在Eclipse创建用户库并加入到构建路径中。

# log4j架构

log4j API设计为分层结构，其中每一层提供了不同的对象，对象执行不同的任务。这使得设计灵活，根据将来需要来扩展。

有两种类型可用在log4j的框架对象。

1. 核心对象： 框架的强制对象和框架的使用。
2. 支持对象： 框架和支持体核心对象，可选的对象执行另外重要的任务。

* **核心对象**

**Logger对象：**

顶级层的Logger，它提供Logger对象。Logger对象负责捕获日志信息及它们存储在一个空间的层次结构。

**布局对象：**

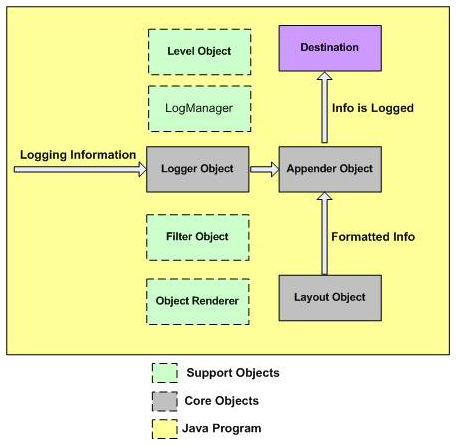
该层提供其用于格式化不同风格的日志信息的对象。布局层提供支持Appender对象到发布日志信息之前。

布局对象的发布方式是人类可读的及可重复使用的记录信息的一个重要的角色。

**Appender对象：**

下位层提供Appender对象。Appender对象负责发布日志信息，以不同的首选目的地，如数据库，文件，控制台，UNIX系统日志等。

以下是显示log4j框架的不同组件的虚拟图：



* **支持对象**：

log4j框架的其他重要的对象起到日志框架的一个重要作用：

**Level对象：**

级别对象定义的任何记录信息的粒度和优先级。有记录的七个级别在API中定义：OFF, DEBUG, INFO, ERROR, WARN, FATAL 和 ALL

**Filter对象**：

过滤对象用于分析日志信息及是否应记录或不用这些信息做出进一步的决定。

一个appender对象可以有与之关联的几个Filter对象。如果日志记录信息传递给特定Appender对象，都和特定Appender相关的Filter对象批准的日志信息，然后才能发布到所连接的目的地。

**对象渲染器：**

ObjectRenderer对象是一个指定提供传递到日志框架的不同对象的字符串表示。这个对象所使用的布局对象来准备最后的日志信息。

**日志管理：**

日志管理对象管理的日志框架。它负责从一个系统级的配置文件或配置类读取初始配置参数。

# log4j配置

配置log4j涉及分配级别，定义追加程序，并在配置文件中指定布局的对象。

log4j.properties文件是一个键 - 值对保存 log4j 配置属性文件。默认情况下，日志管理在CLASSPATH 查找一个名为 log4j.properties 的文件。

1. 根日志记录器的级别定义为DEBUG并连接附加器命名为X到它
2. 设置名为X的附加目的地是一个有效的appender
3. 设置布局的附加器X

* **log4j.properties 语法**

以下是 log4j.properties 文件的一个appender X的语法：

# 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**

使用上面的语法，我们定义 log4j.properties 文件如下：

1. 根日志记录器(logger)的级别定义为DEBUG并连接附加器命名为FILE
2. 附加器(appender)File是定义为org.apache.log4j.FileAppender并写入到一个名为“log.out”位于日志log目录下
3. 定义的布局模式是%m%n，这意味着每打印日志消息之后，将加上一个换行符

# 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

需要注意的是log4j支持UNIX风格的变量替换，如 ${variableName}.

* **调试级别**

使用DEBUG两个追加程序。所有可能的选项有：

1. TRACE
2. DEBUG
3. INFO
4. WARN
5. ERROR
6. FATAL
7. ALL

* **Appenders**

Apache的log4j提供Appender对象主要负责打印日志消息到不同的目的地，如控制台，文件，sockets，NT事件日志等等。

每个Appender对象具有与之相关联的不同的属性，并且这些属性表明对象的行为

|  |  |
| --- | --- |
| 属性 | 描述 |
| layout | Appender使用布局Layout 对象和与之相关的格式化的日志记录信息转换模式 |
| target | 目标可以是一个控制台，一个文件，或根据附加器的另一个项目 |
| level | 级别是必需的，以控制日志消息的过滤 |
| threshold | Appender可以有与之独立的记录器级别相关联的级别阈值水平。Appender忽略具有级别低于阈级别的任何日志消息 |
| filter | Filter 对象可以分析超出级别的匹配记录信息，并决定是否记录的请求应该由一个特定 Appender 或忽略处理 |

可以通过包括以下方法的配置文件中的下面设置一个 Appender 对象添加到记录器：

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

可以编写以XML格式相同的结构如下：

<logger name="com.apress.logging.log4j" additivity="false">

<appender-ref ref="appender1"/>

<appender-ref ref="appender2"/>

</logger>

如果想要添加Appender对象到程序，那么可以使用下面的方法：

public void addAppender(Appender appender);

addAppender()方法添加一个appender到Logger对象。作为示例配置演示，可以添加很多Appender对象到记录器在逗号分隔的列表，每个打印日志信息分离目的地。

我们仅使用一个附加目的地FileAppender在我们上面的例子。所有可能的附加目的地选项有：

1. AppenderSkeleton
2. AsyncAppender
3. ConsoleAppender
4. DailyRollingFileAppender
5. ExternallyRolledFileAppender
6. FileAppender
7. JDBCAppender
8. JMSAppender
9. LF5Appender
10. NTEventLogAppender
11. NullAppender
12. RollingFileAppender
13. SMTPAppender
14. SocketAppender
15. SocketHubAppender
16. SyslogAppender
17. TelnetAppender
18. WriterAppender

我们将涵盖 FileAppender文件 和 JDBCAppender 记录将被包括记录在数据库

* **Layout**

我们使用的PatternLayout 使用 appender。所有可能的选项有：

1. DateLayout
2. HTMLLayout
3. PatternLayout
4. SimpleLayout
5. XMLLayout

使用HTMLLayout和XMLLayout，可以在HTML和XML格式和生成日志。

# log4j示例程序

前面我们已经看到了如何创建一个配置文件。本教程将讲解如何生成调试信息和日志在一个简单的文本文件。

下面是我们的例子中创建了一个简单的配置文件。这里再重复一次：

1. 下载最新的Log4j库：http://logging.apache.org/log4j/2.x/download.html
2. 根记录器的级别定义为DEBUG并连接appender命名为FILE。
3. appender FILE文件被定义为 org.apache.log4j.FileAppender 并写入到一个名为“log.out”位于 log 目录下。
4. 定义的布局模式是 %m%n，这意味着打印日志消息之后自动加上一个换行符。

所以 log4j.properties 文件的内容如下：

# 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

* **在Java程序中使用log4j**

下面的Java类是一个非常简单的例子，Java应用程序初始化，然后使用Log4J日志库。

**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 an debug message");

*log*.info("Hello this is an info message");

}

}

* **编译和运行**

下面是步骤编译并运行上述程序。确保在进行编译和执行之前，适当地设置PATH和CLASSPATH。

所有的库应该在 CLASSPATH 和 log4j.properties 文件应该在PATH可用。所以，做到以下几点：

1. 创建log4j.properties如上图所示。
2. 创建log4jExample.java如上图所示，并对其进行编译。
3. 执行log4jExample二进制运行程序。

在里面 /usr/home/log4j/log.out 文件会得到下面的结果：

Hello this is an debug message

Hello this is an info message

# log4j Logger方法

Logger类提供了多种方法来处理日志活动。 Logger类不允许实例化一个新的记录器实例，但它提供了两个静态方法获得一个 Logger 对象：

1. public static Logger getRootLogger();
2. public static Logger getLogger(String name);

此处两种方法的第一个返回应用程序实例根记录器并没有名字。任何其他命名的Logger对象实例是通过第二种方法通过记录器的名称获得。记录器名称是可以传递任何字符串，通常是类或包的名称，因为我们已经使用在最后一章。

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

* **Logging 方法**

我们得到了一个名为记录器的实例之后，可以使用记录的几种方法来记录消息。 Logger类有专门用于打印日志信息下面的方法如下。

|  |  |
| --- | --- |
| 序号 | 方法及描述 |
| 1 | public void debug(Object message)  这种方法打印使用 Level.DEBUG 消息级别 |
| 2 | public void error(Object message)  这种方法打印使用 Level.ERROR 消息级别 |
| 3 | public void fatal(Object message);  这种方法打印使用 Level.FATAL 消息级别 |
| 4 | public void info(Object message);  这种方法打印使用 Level.INFO 消息级别 |
| 5 | public void warn(Object message);  这种方法打印使用 Level.WARN 消息级别 |
| 6 | public void trace(Object message);  这种方法打印使用Level.TRACE消息级别 |

所有的级别定义在org.apache.log4j.Level类中，并且任何上述方法都可以调用如下：

**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!");

}

}

当编译并运行LogClass程序会产生以下结果：

Debug Message!

Info Message!

Warn Message!

Error Message!

Fatal Message!

所有的调试消息更有意义，当它们在级别组合使用。级别将在下一章介绍，那么在下一节会有一个很好的理解及如何使用这些方法在不同的级别调试。

# log4j日志记录级别

org.apache.log4j.Level类提供以下级别，但也可以通过Level类的子类自定义级别。

|  |  |
| --- | --- |
| Level | 描述 |
| ALL | 各级包括自定义级别 |
| DEBUG | 指定细粒度信息事件是最有用的应用程序调试 |
| ERROR | 错误事件可能仍然允许应用程序继续运行 |
| FATAL | 指定非常严重的错误事件，这可能导致应用程序中止 |
| INFO | 指定能够突出在粗粒度级别的应用程序运行情况的信息的消息 |
| OFF | 这是最高等级，为了关闭日志记录 |
| TRACE | 指定细粒度比DEBUG更低的信息事件 |
| WARN | 指定具有潜在危害的情况 |

* **日志级别是如何工作**

级别p的级别使用q，在记录日志请求时，如果p>=q启用。这条规则是log4j的核心。它假设级别是有序的。对于标准级别它们关系如下：ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF。

下面的例子明确指出如何可以过滤所有的DEBUG和INFO消息。这个程序使用记录并执行setLevel（Level.X）方法来设置所需的日志记录级别：

这个例子将打印，除了调试和信息的所有消息：

**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!");

}

}

当编译并运行LogClass程序会产生以下结果：

Warn Message!

Error Message!

Fatal Message!

* **使用配置文件设置级别**

Log4j提供这些可以让程序员自由更改源代码，改变调试级别的配置级别是基于文件设置。

以下是上面的例子使用 log.setLevel（Level.WARN）方法的配置文件与上面的例子例子功能一样。

# 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，使用下面的程序：

**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!");

}

}

现在，编译和运行上面的程序，得到以下结果在 /usr/home/log4j/log.out 文件：

Warn Message!

Error Message!

Fatal Message!

# log4j日志格式化

Apache log4j 提供了各种布局对象，每一个对象都可以根据各种布局格式记录数据。另外，也可以创建一个布局对象格式化测井数据中的特定应用的方法。

所有的布局对象 - Appender对象收到 LoggingEvent 对象。布局对象检索来自LoggingEvent 的消息参数，并应用适当的 ObjectRenderer 获得消息的字符串表示。

* **布局类型**

在层次结构中的顶级类是抽象类是org.apache.log4j.Layout。这是 log4j 的 API 中的所有其他布局类的基类。

布局类定义为抽象在应用程序中，不要直接使用这个类; 相反，使用它的子类来工作，如下：

1. DateLayout
2. HTMLLayout
3. PatternLayout
4. SimpleLayout
5. XMLLayout

* **布局方法**

这个类提供了一个框架实现在所有其它布局对象的所有常见的操作，并声明了两个抽象方法。

|  |  |
| --- | --- |
| 序号 | 方法及描述 |
| 1 | public abstract boolean ignoresThrowable()  这种方法表示日志信息是否处理传递给它的日志记录事件的一部分，任何 java.lang.Throwable 对象。如果布局对象处理 Throwable 对象，那么布局对象不忽视它，并返回false。 |
| 2 | public abstract String format(LoggingEvent event)  独特的布局子类将实施这一方法的布局特定的格式 |

除了这些抽象方法，布局类提供具体的实现下列方法：

|  |  |
| --- | --- |
| 序号 | 方法及描述 |
| 1 | public String getContentType()  返回使用的布局的对象的内容类型。基类将返回 text/plain 作为默认的内容类型 |
| 2 | public String getFooter()  指定日志消息的页脚信息 |
| 3 | public String getHeader()  指定日志消息的标头信息 |

每个子类可以通过重写的具体实现这些方法返回类特定的信息。

# log4j HTMLLayout

如果想生成一个HTML格式的文件，日志信息，那么可以使用 org.apache.log4j.HTMLLayout 格式化日志信息。

HTMLLayout类扩展抽象org.apache.log4j.Layout类，并覆盖其基类的 format()方法来提供HTML样式格式。

这提供了以下信息显示：

1. 生成特定的日志事件之前，从应用程序的开始所经过的时间
2. 调用该记录请求的线程的名称
3. 与此记录请求相关联的级别
4. 日志记录器(Logger)和记录消息的名称
5. 可选程序文件的位置信息，并从其中记录被调用的行号

HTMLLayout是一个非常简单的布局对象，它提供以下方法：

|  |  |
| --- | --- |
| 序号 | 方法及描述 |
| 1 | setContentType(String)  设置 text/html 为 HTML内容的内容类型。默认为 text/html |
| 2 | setLocationInfo(String)  设置位置信息记录事件。默认为 false |
| 3 | setTitle(String)  设置为HTML文件的标题。默认值是Log4j的日志信息 |

* **HTMLLayout 例子**

以下是对HTMLLayout一个简单的配置文件

# 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

现在考虑下面的Java例子用于产生日志信息

**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");

}

}

编译并运行上述程序，它会在 /usr/home/log4j 目录创建 htmlLayout.html 文件，该文件将有如下的日志信息：

Log session start time Mon Mar 22 13:30:24 AST 2014



可以使用一个Web浏览器打开htmlLayout.html 文件。同样重要的是要注意，页脚</ HTML>和</ body>标记是完全缺失。

一个具有HTML格式的日志文件的一大优势是，它可以被发布为网页可以远程查看。

# log4j PatternLayout

如果想生成基于模式的特定格式的日志信息，那么可以使用 org.apache.log4j.PatternLayout 格式化日志信息。

PatternLayout类扩展抽象 org.apache.log4j.Layout 类并覆盖format()方法根据提供的模式构建日志信息。

PatternLayout也是一个简单的布局对象，它提供下列Bean属性，可以通过配置文件进行设置：

|  |  |
| --- | --- |
| 序号 | 属性和说明 |
| 1 | conversionPattern  设置转换模式。默认为 %r [%t] %p %c %x - %m%n |

* **模式转换字符**

下表说明了以上模式使用的字符和所有其他字符，可以在自定义模式中使用：

|  |  |
| --- | --- |
| 转换字符 | 表示的意思 |
| c | 用于输出的记录事件的类别。例如，对于类别名称"a.b.c" 模式 %c{2} 会输出 "b.c" |
| C | 用于输出呼叫者发出日志请求的完全限定类名。例如，对于类名 "org.apache.xyz.SomeClass", 模式 %C{1} 会输出 "SomeClass". |
| d | 用于输出的记录事件的日期。例如， %d{HH:mm:ss,SSS} 或 %d{dd MMM yyyy HH:mm:ss,SSS}. |
| F | 用于输出被发出日志记录请求，其中的文件名 |
| l | 用于将产生的日志事件调用者输出位置信息 |
| L | 用于输出从被发出日志记录请求的行号 |
| m | 用于输出使用日志事件相关联的应用程序提供的消息 |
| M | 用于输出发出日志请求所在的方法名称 |
| n | 输出平台相关的行分隔符或文字 |
| p | 用于输出的记录事件的优先级 |
| r | 用于输出毫秒从布局的结构经过直到创建日志记录事件的数目 |
| t | 用于输出生成的日志记录事件的线程的名称 |
| x | 用于与产生该日志事件的线程相关联输出的NDC（嵌套诊断上下文） |
| X | 在X转换字符后面是键为的MDC。例如 X{clientIP} 将打印存储在MDC对键clientIP的信息 |
| % | 文字百分号 %%将打印％标志 |

* **格式修饰符**

默认情况下，相关资料原样输出。然而，随着格式修饰符的帮助下，可以改变最小字段宽度，最大字段宽度和对齐。

下表涵盖了各种各样的修饰符的情况：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Format modifier | left justify | minimum width | maximum width | 注释 |
| %20c | false | 20 | none | 用空格左垫，如果类别名称少于20个字符长 |
| %-20c | true | 20 | none | 用空格右垫，如果类别名称少于20个字符长 |
| %.30c | NA | none | 30 | 从开始截断，如果类别名称超过30个字符长 |
| %20.30c | false | 20 | 30 | 用空格左侧垫，如果类别名称短于20个字符。但是，如果类别名称长度超过30个字符，那么从开始截断。 |
| %-20.30c | true | 20 | 30 | 用空格右侧垫，如果类别名称短于20个字符。但是，如果类别名称长度超过30个字符，那么从开始截断。 |

* **PatternLayout 示例**

# 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

现在考虑下面产生日志信息的Java例子：

**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，它会创建 log.out文件在 /usr/home/log4j 目录，该文件将有如下的日志信息：

2010-03-23-main--DEBUG-log4jExample:Hello this is an debug message

2010-03-23-main--INFO -log4jExample:Hello this is an info message

# log4j日志记录到文件

要写日志信息到一个文件中，必须使用org.apache.log4j.FileAppender。有以下FileAppender的配置参数：

* **FileAppender配置：**

|  |  |
| --- | --- |
| 属性 | 描述 |
| immediateFlush | 标志的默认设置为true，这意味着输出流的文件被刷新，在每个追加操作 |
| encoding | 它可以使用任何字符编码。默认情况下是特定于平台的编码方案 |
| threshold | 这个 appender 阈值级别 |
| Filename | 日志文件的名称 |
| fileAppend | 默认设置为true，这意味着记录的信息被附加到同一文件的末尾 |
| bufferedIO | 此标志表示是否需要写入缓存启用。默认设置为false |
| bufferSize | 如果 bufferedI/O 启用，这表示缓冲区的大小，默认设置为8KB |

下面是一个示例配置文件 log4j.properties 的 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

如果喜欢相当于上述log4j.properties文件的XML配置文件，在这里是xml配置文件的内容：

<?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>

* **日志记录到多个文件**

当想要写日志信息转化多个文件要求一样，例如，如果文件大小达到一定的阈值等。

写日志记录信息分成多个文件，必须扩展FileAppender类，并继承其所有属性useorg.apache.log4j.RollingFileAppender类。

有以下除了已如上所述为 FileAppender 可配置参数：

|  |  |
| --- | --- |
| 属性 | 描述 |
| maxFileSize | 上述的文件的回滚临界尺寸。默认值是10MB |
| maxBackupIndex | 此属性表示要创建的备份文件的数量。默认值是1 |

下面是一个示例配置文件log4j.properties的RollingFileAppender进行

# 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

如果想有一个XML配置文件，可以生成中提到的初始段，并添加相关的 RollingFileAppender 进行唯一额外的参数。

此示例配置说明每个日志文件的最大允许大小为5MB。当超过最大尺寸，新的日志文件将被创建并因为maxBackupIndex被定义为2，当第二个日志文件达到最大值，第一个日志文件将被删除，之后所有的日志信息将被回滚到第一个日志文件。

* **每天生成日志文件**

当想生成每一天的日志文件，以保持日志记录信息的良好记录。

日志记录信息纳入日常的基础文件，就必须它扩展FileAppender类，并继承其所有属性useorg.apache.log4j.DailyRollingFileAppender类。

有除了已如上所述为 FileAppender 只有一个重要的下列配置参数：

|  |  |
| --- | --- |
| 属性 | 描述 |
| DatePattern | 这表示在滚动的文件，并按命名惯例来执行。默认情况下，在每天午夜滚动 |

DatePattern控制使用下列滚动的时间表方式之一：

|  |  |
| --- | --- |
| 属性 | 描述 |
| '.' yyyy-MM | 滚动在每个月的结束和下一个月初 |
| '.' yyyy-MM-dd | 这是默认值，每天午夜滚动 |
| '.' yyyy-MM-dd-a | 滚动每一天的午夜和中午 |
| '.' yyyy-MM-dd-HH | 滚动在每一个小时 |
| '.' yyyy-MM-dd-HH-mm | 滚动在每一个分钟 |
| '.' yyyy-ww | 滚动每个星期取决于区域设置时的第一天 |

下面是一个示例配置文件log4j.properties生成日志文件滚动的在每天午夜。

# 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

如果想使用XML配置文件，可以生成中提到的初始段，并添加相关DailyRollingFileAppender 唯一的额外参数和数据。

# log4j日志记录到数据库

log4j API提供 org.apache.log4j.jdbc.JDBCAppender 对象，它能够将日志信息在指定的数据库。

* **JDBCAppender 配置**

|  |  |
| --- | --- |
| 属性 | 描述 |
| bufferSize | 设置缓冲区的大小。默认大小为1 |
| driver | 设置驱动程序类为指定的字符串。如果没有指定驱动程序类，默认为sun.jdbc.odbc.JdbcOdbcDriver |
| layout | 设置要使用的布局。默认布局是org.apache.log4j.PatternLayout |
| password | 设置数据库密码。 |
| sql | 指定SQL语句在每次记录事件发生的时间执行。这可能是INSERT，UPDATE或DELETE |
| URL | 设置JDBC URL |
| user | 设置数据库用户名 |

* **日志表配置**

开始使用基于JDBC日志，要创建在哪里保存日志信息的表。下面是创建日志表的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

);

* **配置文件示例**

以下是将用于将消息记录到一个日志表中的示例配置文件 log4j.properties的JDBCAppender

# 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

这里使用的是MySQL数据库，必须要使用实际DBNAME，用户ID和在其中创建的日志表的数据库密码。SQL语句是使用日志表名和输入值到表，需要执行INSERT语句。

JDBCAppender不需要明确定义的布局。相反，使用PatternLayout 传递给它 SQL语句

如果想拥有相当于上述log4j.properties文件的XML配置文件，可以参考在这里的内容

<?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**

下面的Java类是一个非常简单的Java应用程序使用Log4J日志库例子，初始化，然后使用。

**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");

}

}

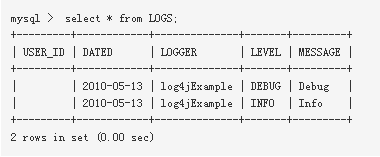
* **编译和运行**

下面是步骤编译并运行上述程序。确保进行编译和执行之前，适当地设置PATH和CLASSPATH。

所有的库应该在CLASSPATH以及log4j.properties文件应该在PATH可用。所以有以下几点：

1. 创建log4j.properties如上图所示。
2. 创建log4jExample.java如上图所示，并对其进行编译。
3. 执行log4jExample二进制运行程序。

现在检查DBNAME数据库里面日志表，发现下面的条目（记录）：



注：此处X被用于产生该记录事件的线程相关联输出的NDC（嵌套诊断上下文）。使用NDC来区分客户的服务器端组件处理多个客户端。检查Log4J的手册以获取更多信息。