**[log4j教程](http://www.blogjava.net/hellxoul/archive/2011/11/30/365181.html)**

1. 简介

2. 安装

3. log4j基本概念

3.1. Logger

3.2. Appender

3.2.1. 使用ConsoleAppender

3.2.2. 使用FileAppender

3.2.3. 使用WriterAppender

3.3. Layout

3.4. 基本示例

3.4.1. SimpleLayout和FileAppender

3.4.2. HTMLLayout和WriterAppender

3.4.3. PatternLayout和ConsoleAppender

4. 使用外部配置文件

5. 参考资料 (以及一些有参考价值的链接)

**1.简介**

程序开发环境中的日志记录是由嵌入在程序中以输出一些对开发人员有用信息的语句所组成。例如，跟踪语句（trace），结构转储和常见的 System.out.println或printf调试语句。log4j提供分级方法在程序中嵌入日志记录语句。日志信息具有多种输出格式和多个输出级别。

使用一个专门的日志记录包，可以减轻对成千上万的System.out.println语句的维护成本，因为日志记录可以通过配置脚本在运行时得以控制。 log4j维护嵌入在程序代码中的日志记录语句。通过规范日志记录的处理过程，一些人认为应该鼓励更多的使用日志记录并且获得更高程度的效率。

**2.安装**

为了使用我们即将要安装的日志记录工具，还必须要设置操作环境，只有这样，工具才能知道从哪里找到它所需要的信息，并且操作系统知道从哪里找到这个工具。那么，怎样做呢？实际上，它要求更改操作环境。

1. 从 <http://jakarta.apache.org/log4j/docs/download.html>下载log4j发行版。
2. 解压存档文件到合适的目录中。
3. 添加文件 dist/lib/log4j-1.2.6.jar 到 CLASSPATH 环境变量中。

**3.log4j的基本概念**

使用log4j大概涉及3个主要概念:

1. 公共类 Logger

Logger 负责处理日志记录的大部分操作。

1. 公共接口 Appender

*Appender* 负责控制日志记录操作的输出。

1. 公共抽象类Layout

Layout 负责格式化Appender的输出。

**3.1.Logger**

日志记录器(Logger)是日志处理的核心组件。log4j具有5种正常级别(Level)。 日志记录器(Logger)的可用级别Level (不包括自定义级别 Level)， 以下内容就是摘自log4j API (<http://jakarta.apache.org/log4j/docs/api/index.html>):

* static Level DEBUG

DEBUG Level指出细粒度信息事件对调试应用程序是非常有帮助的。

* static Level INFO

INFO level表明 消息在粗粒度级别上突出强调应用程序的运行过程。

* static Level WARN

WARN level表明会出现潜在错误的情形。

* static Level ERROR

ERROR level指出虽然发生错误事件，但仍然不影响系统的继续运行。

* static Level FATAL

FATAL level指出每个严重的错误事件将会导致应用程序的退出。

另外，还有两个可用的特别的日志记录级别: (以下描述来自log4j API<http://jakarta.apache.org/log4j/docs/api/index.html>):

* static Level ALL

ALL Level是最低等级的，用于打开所有日志记录。

* static Level OFF

OFF Level是最高等级的，用于关闭所有日志记录。

日志记录器（Logger）的行为是分等级的。

日志记录器（Logger）将只输出那些级别高于或等于它的级别的信息。如果没有设置日志记录器（Logger）的级别，那么它将会继承最近的祖先的级别。因此，如果在包com.foo.bar中创建一个日志记录器（Logger）并且没有设置级别，那它将会继承在包com.foo中创建的日志记录器（Logger）的级别。如果在com.foo中没有创建日志记录器（Logger）的话,那么在com.foo.bar中创建的日志记录器（Logger）将继承root 日志记录器（Logger）的级别，root日志记录器（Logger）经常被实例化而可用，它的级别为DEBUG。

有很多方法可以创建一个日志记录器（Logger），下面方法可以取回root日志记录器:

|  |
| --- |
| Logger logger = Logger.getRootLogger(); |

还可以这样创建一个新的日志记录器:

|  |
| --- |
| Logger logger = Logger.getLogger("MyLogger"); |

比较常用的用法，就是根据类名实例化一个静态的全局日志记录器:

|  |
| --- |
| static Logger logger = Logger.getLogger(test.class); |

所有这些创建的叫"logger"的日志记录器都可以用下面方法设置级别:

|  |
| --- |
| logger.setLevel((Level)Level.WARN); |

可以使用7个级别中的任何一个; Level.DEBUG, Level.INFO, Level.WARN, Level.ERROR, Level.FATAL, Level.ALL and Level.OFF.

**3.2.Appender**

Appender 控制日志怎样输出。下面列出一些可用的Appender(log4j API中所描述的[http://jakarta.apache.org/log4j/docs/api/index.html](http://jakarta.apache.org/log4j/docs/api/index.html" \t "_top)):

1. ConsoleAppender:使用用户指定的布局(layout) 输出日志事件到System.out或者 System.err。默认的目标是System.out。
2. DailyRollingFileAppender 扩展FileAppender，因此多个日志文件可以以一个用户选定的频率进行循环日志记录。
3. FileAppender 把日志事件写入一个文件
4. RollingFileAppender 扩展FileAppender备份容量达到一定大小的日志文件。
5. WriterAppender根据用户的选择把日志事件写入到Writer或者OutputStream。
6. SMTPAppender 当特定的日志事件发生时，一般是指发生错误或者重大错误时，发送一封邮件。
7. SocketAppender 给远程日志服务器（通常是网络套接字节点）发送日志事件（LoggingEvent）对象。
8. SocketHubAppender 给远程日志服务器群组（通常是网络套接字节点）发送日志事件（LoggingEvent）对象。
9. SyslogAppender给远程异步日志记录的后台精灵程序(daemon)发送消息。
10. TelnetAppender 一个专用于向只读网络套接字发送消息的log4j appender。

还可以实现 *Appender* 接口，创建以自己的方式进行日志输出的Appender。

**3.2.1.使用ConsoleAppender**

ConsoleAppender可以用这种方式创建：

|  |
| --- |
| ConsoleAppender appender = new ConsoleAppender(new PatternLayout()); |

创建了一个控制台appender，具有一个默认的PatternLayout。它使用了默认的System.out 输出。

**3.2.2.使用FileAppender**

FileAppender可以用这种方式创建:

|  |
| --- |
| FileAppender appender = null;  try {  appender = new FileAppender(new PatternLayout(),"filename");  } catch(Exception e) {} |

上面用到的构造函数:

|  |
| --- |
| FileAppender(Layout layout, String filename)  实例化一个FileAppender并且打开变量"filename"指定的文件。 |

另一个有用的构造函数是：

|  |
| --- |
| FileAppender(Layout layout, String filename, boolean append)  实例化一个FileAppender并且打开变量"filename"指定的文件。 |

这个构造函数还可以选择是否对指定的文件进行追加的方式输出。如果没有指定值，那么默认的方式就是追加。

**3.2.3.使用WriterAppender**

WriterAppender可以用这种方式创建:

|  |
| --- |
| WriterAppender appender = null;  try {  appender = new WriterAppender(new PatternLayout(),new FileOutputStream("filename"));  } catch(Exception e) {} |

这个WriterAppender使用的构造函数带有PatternLayout和OutputStream参数，在这种情况下， FileOutputStream用于向一个文件输出。当然，它还具有其他可用的构造函数。

**3.3.Layout**

Appender必须使用一个与之相关联的 Layout，这样它才能知道怎样格式化它的输出。当前，log4j具有三种类型的Layout:

1. HTMLLayout 格式化日志输出为HTML表格。
2. PatternLayout 根据指定的 *转换模式*格式化日志输出，或者如果没有指定任何转换模式，就使用默认的转换模式。
3. SimpleLayout 以一种非常简单的方式格式化日志输出，它打印级别 Level，然后跟着一个破折号“-“ ，最后才是日志消息。

**3.4.基本示例**

**3.4.1.SimpleLayout和 FileAppender**

这里是一个非常简单的例子，程序实现了SimpleLayout和FileAppender:

|  |
| --- |
| import org.apache.log4j.Level;  import org.apache.log4j.Logger;  import org.apache.log4j.SimpleLayout;  import org.apache.log4j.FileAppender;  public class simpandfile {  static Logger logger = Logger.getLogger(simpandfile.class);  public static void main(String args[]) {  SimpleLayout layout = new SimpleLayout();  FileAppender appender = null;  try {  appender = new FileAppender(layout,"output1.txt",false);  } catch(Exception e) {}  logger.addAppender(appender);  logger.setLevel((Level) Level.DEBUG);  logger.debug("Here is some DEBUG");  logger.info("Here is some INFO");  logger.warn("Here is some WARN");  logger.error("Here is some ERROR");  logger.fatal("Here is some FATAL");  }  } |

你可以下载: simpandfile.java。 还可以查看它的输出： output1.txt.

**3.4.2.HTMLLayout和 WriterAppender**

这里是一个非常简单的例子，程序实现了 HTMLLayout和WriterAppender:

|  |
| --- |
| import java.io.\*;  import org.apache.log4j.Level;  import org.apache.log4j.Logger;  import org.apache.log4j.HTMLLayout;  import org.apache.log4j.WriterAppender;  public class htmlandwrite {  static Logger logger = Logger.getLogger(htmlandwrite.class);  public static void main(String args[]) {  HTMLLayout layout = new HTMLLayout();  WriterAppender appender = null;  try {  FileOutputStream output = new FileOutputStream("output2.html");  appender = new WriterAppender(layout,output);  } catch(Exception e) {}  logger.addAppender(appender);  logger.setLevel((Level) Level.DEBUG);  logger.debug("Here is some DEBUG");  logger.info("Here is some INFO");  logger.warn("Here is some WARN");  logger.error("Here is some ERROR");  logger.fatal("Here is some FATAL");  }  } |

你可以下载: simpandfile.java. 还可以查看它的输出：output1.txt.

**3.4.3.PatternLayout和 ConsoleAppender**

这里是一个非常简单的例子，程序实现了PatternLayout和ConsoleAppender:

|  |
| --- |
| import org.apache.log4j.Level;  import org.apache.log4j.Logger;  import org.apache.log4j.PatternLayout;  import org.apache.log4j.ConsoleAppender;  public class consandpatt {  static Logger logger = Logger.getLogger(consandpatt.class);  public static void main(String args[]) {  // Note, %n is newline  String pattern = "Milliseconds since program start: %r %n";  pattern += "Classname of caller: %C %n";  pattern += "Date in ISO8601 format: %d{ISO8601} %n";  pattern += "Location of log event: %l %n";  pattern += "Message: %m %n %n";  PatternLayout layout = new PatternLayout(pattern);  ConsoleAppender appender = new ConsoleAppender(layout);  logger.addAppender(appender);  logger.setLevel((Level) Level.DEBUG);  logger.debug("Here is some DEBUG");  logger.info("Here is some INFO");  logger.warn("Here is some WARN");  logger.error("Here is some ERROR");  logger.fatal("Here is some FATAL");  }  } |

你可以下载:simpandfile.java. 还可以查看它的输出： output2.txt.

**4.使用外部配置文件**

Log4j经常与外部日志文件联合使用，这样很多可选项不必硬编码在软件中。使用外部配置文件的优点就是修改可选项不需要重新编译程序。唯一的缺点就是，由于用到*io* 指令，速度稍微有些减慢。

有两个方法可以用来指定外部配置文件：文本文件或者XML文件。既然现在所有事情都写成XML文件，那么该教程就重点讲解XML文件方法，但是也包含相关文本文件的例子。首先，看看下面的XML配置文件示例：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">  <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">  <appender name="ConsoleAppender" class="org.apache.log4j.ConsoleAppender">  <layout class="org.apache.log4j.SimpleLayout"/>  </appender>  <root>  <priority value ="debug" />  <appender-ref ref="ConsoleAppender"/>  </root>  </log4j:configuration> |

文件以标准的XML声明作为开始，后面跟着指出DTD（文档类型定义）的DOCTYPE声明，它定义了XML文件的结构，例如，什么元素可以嵌入在其他元素中等等。上面文件在log4j发行版的src/java/org/apache/log4j/xml目录中。 接着看看封装所有元素的 log4j:configuration 元素，它在DOCTYPE声明中被指定为根元素。嵌入在根元素中有两个结构：

|  |
| --- |
| <appender name="ConsoleAppender" class="org.apache.log4j.ConsoleAppender">  <layout class="org.apache.log4j.SimpleLayout"/>  </appender> |

这里创建一个名叫"ConsoleAppender"的 *Appender*，注意，你可以选择任何名字，该示例之所以选择"ConsoleAppender"，完全是为了示例的设计。接着这个appender类以全名形式给出，经常用规范（fully qualified）类名。 *Appender*必须具有一个指定的 *name*和*class*。嵌入在 *Appender*之内的是 *layout*元素，这里它被指定为SimpleLayout。 *Layout* 必须具有一个 *class*属性。

|  |
| --- |
| <root>  <priority value ="debug" />  <appender-ref ref="ConsoleAppender"/>  </root> |

root元素必须存在且不能被子类化。示例中的优先级被设置为"debug"，设置appender饱含一个*appender-ref*元素。还有更多的属性或元素可以指定。查看log4j发行版中的src/java/org/apache/log4j/xml/log4j.dtd以了解关于XML配置文件结构的更多信息。可以用下面这种方法把配置信息文件读入到Java程序中:

|  |
| --- |
| DOMConfigurator.configure("configurationfile.xml"); |

*DOMConfigurator* 用一棵DOM树来初始化log4j环境。这里是示例中的XML配置文件：plainlog4jconfig.xml。这里是执行该配置文件的程序: files/externalxmltest.java:

|  |
| --- |
| import org.apache.log4j.Logger;  import org.apache.log4j.xml.DOMConfigurator;  public class externalxmltest {  static Logger logger = Logger.getLogger(filetest.class);  public static void main(String args[]) {  DOMConfigurator.configure("xmllog4jconfig.xml");  logger.debug("Here is some DEBUG");  logger.info("Here is some INFO");  logger.warn("Here is some WARN");  logger.error("Here is some ERROR");  logger.fatal("Here is some FATAL");  }  } |

这里是一个实现带有*PatternLayout*的*FileAppender*的日志记录器Logger的XML配置文件:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">  <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">  <appender name="appender" class="org.apache.log4j.FileAppender">  <param name="File" value="Indentify-Log.txt"/>  <param name="Append" value="false"/>  <layout class="org.apache.log4j.PatternLayout">  <param name="ConversionPattern" value="%d [%t] %p - %m%n"/>  </layout>  </appender>  <root>  <priority value ="debug"/>  <appender-ref ref="appender"/>  </root>  </log4j:configuration> |

你可以从这里下载示例: xmllog4jconfig2.xml。 想要得到更多的使用XML文件配置log4j环境的例子，请查看log4j发行版的目录src/java/org/apache/log4j/xml/examples/ 。

这就是上面讨论的文本文件形式的配置文件:

|  |
| --- |
| # initialise root logger with level DEBUG and call it BLAHlog4j.rootLogger=DEBUG,  BLAH# add a ConsoleAppender to the logger BLAHlog4j.appender.BLAH=  org.apache.log4j.ConsoleAppender# set set that layout to be SimpleLayoutlog4j.appender.BLAH.layout=  org.apache.log4j.SimpleLayout |

从这里可以下载: plainlog4jconfig.txt。这就是执行该配置文件的程序:

|  |
| --- |
| import org.apache.log4j.Logger;  import org.apache.log4j.PropertyConfigurator;  public class externalplaintest {  static Logger logger = Logger.getLogger(externalplaintest.class);  public static void main(String args[]) {  PropertyConfigurator.configure("plainlog4jconfig.xml");  logger.debug("Here is some DEBUG");  logger.info("Here is some INFO");  logger.warn("Here is some WARN");  logger.error("Here is some ERROR");  logger.fatal("Here is some FATAL");  }  } |

你可以下载使用该配置文件的示例： externalplaintest.java。想要获得更多的使用文本文件配置log4j环境的例子，请查看log4j发行版中的目录examples。

使用外部配置文件的例子就简单的讨论到这里，现在应该可以肯定你已经有能力独立学习更多的log4j发行版和测试版中提供的例子