Log4j使用详解

# 1、介绍

Log4j是Apache的一个开放源代码项目，通过使用Log4j，可以控制日志信息输送的目的地是控制台、文件、网页、甚至是套接口服务器、NT的事件记录器等；也可以控制每一条日志的输出格式。通过定义每一条日志信息的级别，能够更加细致地控制日志的生成过程。这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

Log4j是一个开源的，轻量级的，用于日志管理的框架。

Log4j的三个重要组件：

1. 日志信息输出格式：控制日志显示的内容和样式。
2. 日志信息的优先级：描述日志信息的重要程度。
3. 日志信息输出的目的地：指定日志信息时打印到控制台、文件中还是发送到邮件或者保存到数据库中。

# 2、Log4j的级别

Log4j有如下级别(优先级由低到高)：ALL、DEBUG、INFO、WARN、ERROR、FATAL、OFF

一般建议使用四个级别：DEBUG、INFO、WARN、ERROR。通过定义级别，可以控制日志输出相关级别以上的信息，低于当前定义的优先级的日志不输出。

# 3、输出目的地

Log4j允许日志同时又多个输出目的地，一个输出目的地称作一个Appender。

常见的几个Appender如下：

org.apache.log4j.ConsoleAppender（控制台）

org.apache.log4j.FileAppender（文件）

org.apache.log4j.DailyRollingFileAppender（每天产生一个日志文件）

org.apache.log4j.RollingFileAppender（文件大小到达指定尺寸的时候产生一个新的文件）

org.apache.log4j.WriterAppender（将日志信息以流格式发送到任意指定的地方）

org.apache.log4j.jdbc.JDBCAppender（把日志用JDBC记录到数据库中）

org.apache.log4j.net.SMTPAppender（把日志发送到邮件）

# 4、Log4j的七个核心类、接口

分别是：Logger、LoggerRepository、Level、LoggingEvent、Appender、Layout、ObjectRender

类图如下：



更详细的功能类图：



Log4j Api(<http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/package-tree.html>)的Tree结构：



其实，log4j最核心的就5个类：

Logger： 对日志记录行为的抽象，提供记录不同级别日志的接口;

Level：对日志级别的抽象；

LoggingEvent：对一次日志记录过成功，所能去到信息的抽象；

Appender：对日志记录形式的抽象；

Layout：对日志行格式的抽象。

一次简单的日志记录过程如下：



以上进行了Log4j基本结构和作用流程的介绍，接下来对log4j的使用进行详细介绍。

# 5、Log4j配置文件介绍

## 5.1 定义配置文件

Log4j支持两种格式的配置文件，一种是XML格式文件，一种是java特性的.properties格式文件。本文以.properties文件为例。

1. 定义根Logger，语法如下：

log4j.rootLogger = [Level],appendName1, appendName2, appendName3,…

[Level]是记录日志的优先级，如：ALL、DEBUG、INFO、WARN、ERROR、FATAL、OFF。建议只用四个：DEBUG、INFO、WARN、ERROR

appendName2是指定日志输出的地方。

2）日志输出的目的地Appender，语法如下：

log4j.appender.appenderName = Appender类的全路径

log4j.appender.appenderName.option1 = value1

log4j.appender.appenderName.option2 = value2

…

log4j.appender.appenderName.optionN = valueN

Appender的种类在上文已有介绍。

3）配置日志信息格式，语法为：

log4j.appender.appenderName.layout = Layout类的全路径

log4j.appender.appenderName.option1 = value1

log4j.appender.appenderName.option2 = value2

…

log4j.appender.appenderName.optionN = valueN

Log4j提供的layout有以e几种：

org.apache.log4j.HTMLLayout（以HTML表格形式布局），

org.apache.log4j.PatternLayout（可以灵活地指定布局模式），

org.apache.log4j.SimpleLayout（包含日志信息的级别和信息字符串），

org.apache.log4j.TTCCLayout（包含日志产生的时间、线程、类别等等信息）

Log4j格式化日志的参数及其用法介绍如下：

%p 输出优先级，即DEBUG，INFO，WARN，ERROR等

%r 输出自应用启动到输出该log信息耗费的毫秒数

%c 输出所属的类目，通常就是所在类的全名

%t 输出产生该日志事件的线程名

%n 输出一个回车换行符，Windows平台为“rn”，Unix平台为“n”

%d 输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，比如：%d{yyyy-MM-dd HH:mm:ss,SSS}，输出类似：2012-10-18 14:33:28,921  
　　%l 输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数。举例：MyTestLog4j.main(MyTestLog4j.java:28)

1. 调用Log4j

在JAVA代码中，类里面的第一行定义一个Logger，以类的名字创建，如下：

static Logger logger = Logger.getLogger (MyTestLog4j.class.getName () )

在需要的地方，插入日志记录，调用一下方法（Logger是接口）：

Logger.debug ( Object message ) ;  
Logger.info ( Object message ) ;  
Logger.warn ( Object message ) ;  
Logger.error ( Object message ) ;

## 5.2 示例

导入必要的包

commons-logging-1.1.1.jar

log4j-1.2.15.jar

写一个抛异常的类：

|  |
| --- |
| **public** **class** TestLog4j {  **public** **static** **void** main(String[]args){  **final** Logger log = Logger.*getLogger*(TestLog4j.**class**);  **try**{  System.*out*.println("抛异常：");  Integer.*parseInt*("ad");  **throw** **new** RuntimeException();  }**catch**(Exception e){  //e.printStackTrace();  StringWriter sw = **new** StringWriter();  PrintWriter pw = **new** PrintWriter(sw);  String errorLog = "";  e.printStackTrace(pw);  errorLog = sw.toString(); //获得堆栈信息  log.error(errorLog,e); //调用error方法，记录日志  pw.close();  **try** {  sw.close();  } **catch** (IOException e1) {  e1.printStackTrace();  }  }  }  } |

log4j.properties文件如下：

输出到控制台：

|  |
| --- |
| #这里的Level使用了ERROR，定义了名为CONSOLE的Appender  log4j.rootLogger=ERROR,CONSOLE  #输出控制台  log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender  log4j.appender.CONSOLE.Target=System.out  log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout  log4j.appender.CONSOLE.layout.ConversionPattern=[ErrorMessage] %d - [%t] %-5p %c:%L - %m%n |

输出到日志文件。一下定义了一个名为FILE的Appender，必须在log4j.rootLogger中加入FILE：

|  |
| --- |
| # DailyRollingFileAppender每天输出一个不同的日志文件  log4j.appender.FILE = org.apache.log4j.DailyRollingFileAppender  log4j.appender. FILE.File = d:/logs/error.log  log4j.appender. FILE.Append = true  #只输出ERROR级别以上的日志  log4j.appender. FILE.Threshold = ERROR  log4j.appender. FILE.layout = org.apache.log4j.PatternLayout  log4j.appender. FILE.layout.ConversionPattern = %d - [%t] %-5p %c:%L - %m%n |

日志发送到邮件：定义一个名为EMAIL的Appender，如下：

|  |
| --- |
| #邮件发送  # log4j的邮件发送appender，如果有必要你可以写自己的appender  log4j.appender.MAIL=org.apache.log4j.net.SMTPAppender  #发送邮件的条件，仅当等于或高于ERROR（比如FATAL）时，邮件才被发送  log4j.appender.MAIL.Threshold=ERROR  #邮件缓冲区大小  log4j.appender.MAIL.BufferSize=1024  #发送邮件的邮箱帐号  log4j.appender.MAIL.From=？？？？@163.com  #邮件接收者,群发以英文逗号隔开  log4j.appender.MAIL.To=？？？？@yeah.net  #SMTP邮件发送服务器地址  log4j.appender.MAIL.SMTPHost=smtp.163.com  #SMTP发送认证的帐号名，以“,”分隔，可群发  log4j.appender.MAIL.SMTPUsername=？？？？@163.com  #SMTP发送认证帐号的密码  log4j.appender.MAIL.SMTPPassword=\*\*\*\*\*\*\*\*\*  #是否打印调试信息，如果选true，则会输出和SMTP之间的握手等详细信息  log4j.appender.MAIL.SMTPDebug=true  #邮件主题  log4j.appender.MAIL.Subject=[Log4J ErrorMessage]  log4j.appender.MAIL.layout=org.apache.log4j.PatternLayout  log4j.appender.MAIL.layout.ConversionPattern=[ErrorMessage] %d - [%t] %-5p %c - %m%n |

发送邮件必须导入两个包：

activation.jar

mail-1.4.jar

**注意：**在MyEclipse中第一次使用java发送邮件时，会报“类没有被引用什么什么”的错，是因为引入的mail.jar、activation.jar跟MyEclipse自带的的javaee.jar包冲突,可以把MyEclipse里面的javaee.jar打开删除里面的mail目录和activation目录删除，这样就可以了。

日志保存到数据库，定义一个名为DATABASE的Appender,如下：

|  |
| --- |
| # 写入数据库  log4j.appender.DATABASE=org.apache.log4j.jdbc.JDBCAppender  log4j.appender.DATABASE.URL=jdbc\:odbc\:DRIVER\={Microsoft Access Driver (\*.mdb, \*.accdb)};DBQ\=F\:\\TestDB\\Test.accdb  log4j.appender.DATABASE.driver=sun.jdbc.odbc.JdbcOdbcDriver  log4j.appender.DATABASE.user=  log4j.appender.DATABASE.password=  #log4j.appender.DATABASE.URL=jdbc:mysql://127.0.0.1:3306/test  #log4j.appender.DATABASE.driver=com.mysql.jdbc.Driver  #log4j.appender.DATABASE.user=super  #log4j.appender.DATABASE.password=super  # 本处设置为"ERROR"或以上级别在数据库存储  log4j.appender.DATABASE.Threshold=ERROR  # 存储数据  log4j.appender.DATABASE.sql=INSERT INTO log4j(OccurTime,ThreadName,Priority,ClassName,Message) VALUES ('%d{yyyy-MM-dd HH:mm:ss}','%t', '%-5p','%c:%L','%m%n')  log4j.appender.DATABASE.layout=org.apache.log4j.PatternLayout  log4j.appender.DATABASE.layout.ConversionPattern=%d{yyyy-MM-dd HH\:mm\:ss} - [%t] %-5p %c\:L - %m%n |

创建一个数据表：

|  |
| --- |
| CREATE TABLE `log4j` (  `ID` bigint(20) NOT NULL AUTO\_INCREMENT,  `OccurTime` datetime DEFAULT NULL COMMENT '发生时间',  `ThreadName` varchar(100) DEFAULT NULL COMMENT '线程名',  `Priority` varchar(10) DEFAULT NULL COMMENT '优先级',  `ClassName` varchar(200) DEFAULT NULL COMMENT '类名:行数',  `Message` text COMMENT '异常信息',  PRIMARY KEY (`ID`)  ) ENGINE=InnoDB AUTO\_INCREMENT=59 DEFAULT CHARSET=utf8 COMMENT='运行时的异常信息'; |

注意：

如果不对堆栈信息进行处理，堆栈信息时不会全部存入库的（Message字段）。而只会存一行信息你已经写好的提示信息，在不提示已定义消息的情况下，只会存入一行如下格式的信息：

类名.方法名()

如果要完整滴存入堆栈信息，得做如下处理，把堆栈信息转化成字符串。

|  |
| --- |
| StringWriter sw = **new** StringWriter();  PrintWriter pw = **new** PrintWriter(sw);  String errorLog = "";  e.printStackTrace(pw);  errorLog = sw.toString();  log.error(errorLog,e); |

2012-10-29

Laizhiwen