Log4j 配置

[1. 参考资料 2](#_Toc490226165)

[2. Log4j简介 2](#_Toc490226166)

[2.1. Log4j 2](#_Toc490226167)

[2.2. Log4j优点 2](#_Toc490226168)

[2.3. 为什么需要Log4j 2](#_Toc490226169)

[2.4. Logger.getLogger()和LogFactory.getLog()的区别 3](#_Toc490226170)

[3. common-logging组件 3](#_Toc490226171)

[4. Log4j使用及配置 4](#_Toc490226172)

[4.1. Log4j使用步骤 4](#_Toc490226173)

[4.2. Log4j配置说明 6](#_Toc490226174)

[4.2.1. 定义logger 6](#_Toc490226175)

[4.2.2. 定义appender 7](#_Toc490226176)

[4.2.3. 定义layout 7](#_Toc490226177)

[4.3. ConversionPattern参数的格式含义 8](#_Toc490226178)

[4.4. log4j的配置文件：log4j.properties 8](#_Toc490226179)

[4.5. log4j的日志文件保存位置解决方案 9](#_Toc490226180)

[4.5.1. 绝对路径 9](#_Toc490226181)

[4.5.2. 使用已有jvm变量 9](#_Toc490226182)

[4.5.3. 项目启动时通过System.setProperty设置 9](#_Toc490226183)

[5. Log4j 与架构结合 10](#_Toc490226184)

[5.1. 没有架构的log4j 10](#_Toc490226185)

[5.1.1. log4j.properties 10](#_Toc490226186)

[5.1.2. java代码 12](#_Toc490226187)

[5.2. 使用spring的log4j 14](#_Toc490226188)

[5.3. 使用AOP的log4j 14](#_Toc490226189)

[6. 问题与解决 14](#_Toc490226190)

[6.1. DailyRollingFileAppender不生成昨天日志 14](#_Toc490226191)

# 参考资料

http://m.oschina.net/blog/144218

http://www.blogjava.net/freeman1984/archive/2010/06/10/323236.html

http://blog.csdn.net/xiaogugood/article/details/9366779

# Log4j简介

## Log4j

Log4j 是Apache的一个开放源代码项目，通过使用Log4j，我们可以控制日志信息输送的目的地是控制台、文件、GUI组件、甚至是套接口服务器、NT的事件记录器、UNIX Syslog守护进程等；我们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，我们能够更加细致地控制日志的生成过程。最令人感兴趣的是，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优点

a）Log4j受大多数web应用服务器的拥护，tomcat，weblogic，websphere，jboss等都支持log4j。

b）快速，功能强大：Log4J配置文件实现了输出到控制台、文件、回滚文件、发送日志邮件、输出到数据库日志表、自定义标签等全套功能。log4j在速度上也很快。

c）使用简单、方便：只需要导入一个简单的log4j-1.2.x.jar，然后在程序类的开头写上下面一句private final static Logger log =Logger.getLogger(ClassName.class); 这样你就得到了一个日志对象log，可以轻松往特定目标写日志了。

## 为什么需要Log4j

项目的调试是log4j产生的内在驱动力。

原始的方法是：把信息输出到屏幕（console），利用JDK提供的System.out.println。但是，这样做的坏处是显而易见的：

a）信息的输出不够灵活，并且繁琐。比如，要输出执行处的文件名，行数，当前时间等，println显得很原始。

b）如果要改变输出的内容和格式，需要重新编译源程序。

c）更严重的是，如果程序中有很多的println，会严重的影响程序的性能。

## Logger.getLogger()和LogFactory.getLog()的区别

Logger.getLogger()是使用log4j的方式记录日志；

LogFactory.getLog()则来自apache的common-logging包。

# common-logging组件

Jakarta Commons Logging (JCL)提供的是一个日志(Log)接口(interface)，同时兼顾轻量级和不依赖于具体的日志实现工具。它提供给中间件/日志工具开发者一个简单的日志操作抽象，允许程序开发人员使用不同的具体日志实现工具。Log(基本记录器)和LogFactory(负责创建Log实例)是两个基类。该API直接提供对下列底层日志记录工具的支持：Jdk14Logger，Log4JLogger，LogKitLogger，NoOpLogger （直接丢弃所有日志信息），还有一个SimpleLog。有必要详细说明一下调用LogFactory.getLog()时发生的事情。调用该函数会启动一个发现过程，即找出必需的底层日志记录功能的实现，具体的发现过程在下面列出：(换句话说就是，有这么多工具，common-logging该使用哪一个呢？这取决于系统的设置，common-logging将按以下顺序决定使用哪个日志记录工具)：

(1).common-logging首先在CLASSPATH中查找commons-logging.properties文件。这个属性文件至少定义org.apache.commons.logging.Log属性，它的值应该是上述任意Log接口实现的完整限定名称。如果找到 org.apache.commons.logging.Log属相，则使用该属相对应的日志组件。结束发现过程。

(2).如果上面的步骤失败（文件不存在或属相不存在），common-logging接着检查系统属性org.apache.commons.logging.Log。如果找到org.apache.commons.logging.Log系统属性，则使用该系统属性对应的日志组件。结束发现过程。

(3).如果找不到org.apache.commons.logging.Log系统属性，common-logging接着在CLASSPATH中寻找log4j的类。如果找到了就假定应用要使用的是log4j。不过这时log4j本身的属性仍要通过log4j.properties文件正确配置。结束发现过程。

(4).如果上述查找均不能找到适当的Logging API，但应用程序正运行在JRE 1.4或更高版本上，则默认使用JRE 1.4的日志记录功能。结束发现过程。

(5).最后，如果上述操作都失败（JRE 版本也低于1.4），则应用将使用内建的SimpleLog。SimpleLog把所有日志信息直接输出到System.err。结束发现过程。

为了简化配置 commons-logging ，一般不使用 commons-logging 的配置文件，也不设置与 commons-logging 相关的系统环境变量，而只需将 Log4j 的 Jar 包放置到 classpash 中就可以了。这样就很简单地完成了 commons-logging 与 Log4j 的融合。

# Log4j使用及配置

## Log4j使用步骤

S1先要下载相应的jar包（log4j.jar）。

S2配置web.xml，在web.xml中引入log4j配置文件、log4j的监听器。

<context-param>

<param-name> log4jConfigLocation</param-name>

<param-value>/WEB-INF/props/log4j.properties</param-value>

</context-param>

<context-param>

<param-name> log4jRefreshInterval</param-name>

<param-value>6000</param-value>

</context-param>

<listener>

<listener-class>

org.springframework.web.util.Log4jConfigListener

</listener-class>

</listener>

说明：在上文的配置里，Log4jConfigListener会去WEB- INF/props/log4j.propeties 读取配置文件，开一条watchdog线程每60秒扫描一下配置文件的变化（这样在web服务启动后再去修改配置文件也不用重新启动web服务了），并把 web目录的路径压入一个叫webapp.root的系统变量（webapp.root将在log4j.properties文件中使用）。

S3接下来是log4j.properties配置文件了，把它放在WEB-INF/props下，具体配置如下：

#log4j.rootLogger = [ level ] , appenderName, appenderName, ...

log4j.rootLogger = INFO, console, R

#level=INFO,all can be output

#console is set to be a ConsoleAppender

log4j.appender.console = org.apache.log4j.ConsoleAppender

#console have four patterns

#org.apache.log4j.HTMLLayout

#org.apache.log4j.PatternLayout

#org.apache.log4j.SimpleLayout

#org.apache.log4j.TTCCLayout

log4j.appender.console.layout = org.apache.log4j.PatternLayout

#define the output type

log4j.appender.console.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [%c]-[%p] %m%n

#file is set to output to a extra file

log4j.appender.R = org.apache.log4j.RollingFileAppender

#the absolute route of the log4j file

log4j.appender.R.File = /log.txt

#the size

log4j.appender.R.MaxFileSize = 500KB

#back up a file

log4j.appender.R.MaxBackupIndex = 1

log4j.appender.R.layout = org.apache.log4j.PatternLayout

log4j.appender.R.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss} [%c]-[%p] - %m%n

上面的配置文件说明log信息将以两种方式输出（文件和控制台），表示应用的根目录下（例如本应用名称为ABC，则log.txt的位置为tomact\webapp\ABC下）

S4最后在程序中想要输出log的地方加入log4j的支持

（1）引入 import org.apache.log4j.Logger

（2）声明一个logger

private static Logger logger = Logger.getLogger(ClassName.class);

（3）在程序中的相应位置加入输出信息

logger.info("用户登录:"+user.getAccount());

当有登录时会在控制台和文件中同时输出log信息如下

2007-01-10 16:02:54 [com.my.web.UserAction]-[INFO] 用户登录:yangsq

## Log4j配置说明

Log4j核心包括记录器（Logger），输出端（appenders）和布局（layouts）。

Logger类是日志包的核心，Logger的名称是大小写敏感的，并且名称之间有继承关系。子名由父名做前缀，用点号“.”分隔，如x.y是x.y.z的父亲Logger。Logger系统中有个rootLogger，是所有logger的祖先，它总是存在的，并且不可以通过名字获取，可以通过Logger.getRootLogger()来获取。获取Logger对象的方法很多，可以参考API文档，在某对象中，用该对象所属的类作为参数，调用Logger.getLogger(Class clazz)以获取logger对象被认为是目前所知最理智的命名logger方法。

每个logger都有一个日志级别，用来控制日志的输出。未分配级别的logger将自动继承它最近的父logger的日志级别。Logger的由低到高级别如下：

ALL<DEBUG<INFO<WARN<ERROR<FATAL<OFF

### 定义logger

根记录器与自定义记录器的格式稍微不同，根记录器是log4j.rootLogger，

log4j.rootLogger = [ level ], appendName1, appendName2, …appendNameN

非rootLogger可以使用类名、包名、自定义logger名来设置logger的名称，一个logger可以包含多个appender，一个appender可以被多个logger使用。

包含appender的logger都会自动调用rootLogger，所以rootLogger的appender不必重复。RootLogger也可以有多个appender。

每一个logger都可有多个输出端。

og4j.rootLogger=INFO,Console1,log4j #根记录器，这里设置Console和log4j一共2个appender

log4j.logger.myLogger1=INFO,myFile1 #自定义名的logger，级别为info，比info低级的不显示，一个appender

log4j.logger.myLogger2=debug,myFile2

log4j.logger.log4j.mypk=INFO,mypk #也可以利用包名设置相应的logger

log4j.logger.log4j.mypk.dbpk=debug,dbpk #包名设置logger，会继承上级的appender和根的

log4j.logger.log4j.mypk.MyDb=INFO,mydb #类名设置logger，会继承上级包的appender和根的

level的级别（此级别可以自定义，系统默认提供了以下级别）

◆debug//调试信息

◆info//一般信息

◆warn//警告信息

◆error//错误信息

◆fatal//致命错误信息

上面列出的就是所谓log4j的输出级别，log4j建议只使用4个级别，它们从上到下分别为ERROR、WARN、INFO、DEBUG，假设你定义的级别是info，那么error和warn的日志可以显示而比他低的debug信息就不显示了。

### 定义appender

log4j.appender.appenderName = fully.qualified.name.of.appender.class

log4j提供了以下几种常用的输出目的地：

◆org.apache.log4j.ConsoleAppender，将日志信息输出到控制台

◆org.apache.log4j.FileAppender，将日志信息输出到一个文件

◆org.apache.log4j.DailyRollingFileAppender，将日志信息输出到一个文件，并且每个次日都会生成新的日志文件。注意如果第二天没有日志操作，就不会自动创建昨天的文件，原因在于log4j的设计者考虑24点的时候项目不一定在运行中，所以无法保证每一天都生成。

◆org.apache.log4j.RollingFileAppender，将日志信息输出到一个文件，通过指定文件的的尺寸，当文件大小到达指定尺寸的时候会自动把文件改名，如名为example.log的文件会改名为 example.log.1，同时产生一个新的example.log文件。如果新的文件再次达到指定尺寸，又会自动把文件改名为 example.log.2，同时产生一个example.log文件。依此类推，直到example.log. MaxBackupIndex， MaxBackupIndex的值可在配置文件中定义。

◆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，通过JDBC把日志信息输出到数据库中。

### 定义layout

Log4j提供了一下几种布局：

◆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

定义一个PatternLayout布局的语句为：

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1} - %m%n

## ConversionPattern参数的格式含义

%c 输出日志信息所属的类的全名

%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{yyy-MM-dd HH:mm:ss }，输出类似：2002-10-18- 22：10：28

%f 输出日志信息所属的类的类名

%l 输出日志事件的发生位置，即输出日志信息的语句处于它所在的类的第几行

%m 输出代码中指定的信息，如log(message)中的message

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

%p 输出优先级，即DEBUG，INFO，WARN，ERROR，FATAL。如果是调用debug()输出的，则为DEBUG，依此类推

%r 输出自应用启动到输出该日志信息所耗费的毫秒数

%t 输出产生该日志事件的线程名

## log4j的配置文件：log4j.properties

一个log4j.properties文件示例

log4j.rootLogger=INFO, stdout, logfile

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - <%m>%n

log4j.appender.logfile=org.apache.log4j.RollingFileAppender

log4j.appender.logfile.File=/webserver/specialTraining3/wangzj.log

log4j.appender.logfile.MaxFileSize=51200KB

log4j.appender.logfile.MaxBackupIndex=3

log4j.appender.logfile.layout=org.apache.log4j.PatternLayout

log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - <%m>%n

## log4j的日志文件保存位置解决方案

### 绝对路径

log4j.appender.A1.File=D:/apache-tomcat-6.0.18/webapps/项目/logs/app.log

但这种写法灵活性很差。

### 使用已有jvm变量

log4j.appender.logfile.File=${user.home}/logs/app.log

日志将位于：例如windows：C:\Documents and Settings\joe\logs\app.log

### 项目启动时通过System.setProperty设置

通过实现ServletContextListener来解决：例如

public class log4jlistener implements ServletContextListener {

public static final String log4jdirkey = "log4jdir";

public void contextDestroyed(ServletContextEvent servletcontextevent) {

System.getProperties().remove(log4jdirkey);

}

public void contextInitialized(ServletContextEvent servletcontextevent) {

String log4jdir = servletcontextevent.getServletContext().getRealPath("/");

//System.out.println("log4jdir:"+log4jdir);

System.setProperty(log4jdirkey, log4jdir);

}

}

web.xml配置：

<listener>

<listener-class>com.log4j.log4jlistener</listener-class>

</listener>

log4j.prtperties 配置：

log4j.appender.A1.File=${log4jdir}/WEB-INF/logs/app1.log

# Log4j 与架构结合

不同的架构在使用log4j时一般只有配置文件的差别，如properties、xml、aop等差别。

注意，在本次demo中，可能使用common-logging的包获取logger，使用log4j获取logger实例化对象的方式是相似的，本demo不再区分。这两之间的比较如下：

private static Log logger = LogFactory.getLog(CommonLoggingRootLogger.class);//没有配置.使用rootlogger

private static Logger logger = Logger.getRootLogger();

private static Log logger1 = LogFactory.getLog("myLogger1");//通过名称获取logger

private static Logger logger1 = Logger.getLogger("myLogger1");

在java中获取logger的方法可以使用LogFactory.getLog(“loggerName”)或者LogFactory.getLog(MyDb.class)。

LogFactory.getLog(MyDb.class)会获取到的logger包括rootLogger，以及该类名对应的以及该类所在包的各个父包所对应的logger。

LogFactory.getLog(“loggerName”)会获取loggerName对应的logger，以及rootLogger，可以避免使用类名或者包名获取logger时带来的多层级logger问题。

## 没有架构的log4j

### log4j.properties

(存放在maven项目的resources下即可)

######下面开始定义多个logger

log4j.rootLogger=INFO,Console1,log4j #根记录器，这里设置Console和log4j一共2个appender

log4j.logger.myLogger1=INFO,myFile1 #自定义名的logger，级别为info，比info低级的不显示，一个appender

log4j.logger.myLogger2=debug,myFile2

log4j.logger.log4j.mypk=INFO,mypk #也可以利用包名设置相应的logger

log4j.logger.log4j.mypk.dbpk=debug,dbpk #包名设置logger，会继承上级的appender和根的

log4j.logger.log4j.mypk.MyDb=INFO,mydb #类名设置logger，会继承上级包的appender和根的

#######下面开始定义各个输出流appender

#Console1 appender

log4j.appender.Console1=org.apache.log4j.ConsoleAppender

log4j.appender.Console1.Target=System.out

log4j.appender.Console1.layout = org.apache.log4j.PatternLayout

log4j.appender.Console1.layout.ConversionPattern=[%c] - %m%n

#log4j appender文件大小到达指定尺寸的时候产生一个新的文件

log4j.appender.log4j = org.apache.log4j.RollingFileAppender

log4j.appender.log4j.File = logs/log4j/log4j.log #当前项目根目录下创建路径

log4j.appender.log4j.encoding = UTF-8 #支持linux下中文编码问题

log4j.appender.log4j.MaxFileSize = 10MB

log4j.appender.log4j.Threshold = ALL #过滤器，低于all级别的不显示，用于输出流个性化级别

log4j.appender.log4j.layout = org.apache.log4j.PatternLayout

log4j.appender.log4j.layout.ConversionPattern =[%p] [%d{yyyy-MM-dd HH\:mm\:ss}][%c]%m%n

#myFile1 appender

log4j.appender.myFile1 = org.apache.log4j.DailyRollingFileAppender

log4j.appender.myFile1.File = logs/log4j/myFile1.log

log4j.appender.myFile1.Encoding = UTF-8

log4j.appender.myFile1.DatePattern='\_'yyyy-MM-dd'.log'

log4j.appender.myFile1.Append = true

log4j.appender.myFile1.BufferSize=5k

log4j.appender.myFile1.ImmediateFlush=true #设置立即写入文件

log4j.appender.myFile1.BufferedIO=false

log4j.appender.myFile1.Threshold = info

log4j.appender.myFile1.layout = org.apache.log4j.PatternLayout

log4j.appender.myFile1.layout.ConversionPattern = %m%n

#myFile2 appender

log4j.appender.myFile2 = org.apache.log4j.DailyRollingFileAppender

log4j.appender.myFile2.File = logs/log4j/myFile2.log

log4j.appender.myFile2.Encoding = UTF-8

log4j.appender.myFile2.DatePattern='\_'yyyy-MM-dd'.log'

log4j.appender.myFile2.Append = true

log4j.appender.myFile2.Threshold = info

log4j.appender.myFile2.layout = org.apache.log4j.PatternLayout

log4j.appender.myFile2.layout.ConversionPattern = %m%n

#mypk appender

log4j.appender.mypk=org.apache.log4j.FileAppender

log4j.appender.mypk.File=logs/log4j/mypk.log

log4j.appender.mypk.Append=true

log4j.appender.mypk.layout=org.apache.log4j.PatternLayout

log4j.appender.mypk.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} [%-5p][%c{1}] - %m%n

#dbpk appender

log4j.appender.dbpk=org.apache.log4j.FileAppender

log4j.appender.dbpk.File=logs/log4j/dbpk.log

log4j.appender.dbpk.Append=true

log4j.appender.dbpk.layout=org.apache.log4j.PatternLayout

log4j.appender.dbpk.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} [%-5p][%c{1}] - %m%n

#mydb appender

log4j.appender.mydb=org.apache.log4j.FileAppender

log4j.appender.mydb.File=logs/log4j/mydb.log

log4j.appender.mydb.Append=true

log4j.appender.mydb.layout=org.apache.log4j.PatternLayout

log4j.appender.mydb.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} [%-5p][%c{1}] - %m%n

### java代码

package log4j;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

public class RootLogger {

// 配置中没有包含该类的logger，所以使用rootLogger

private static Log logger = LogFactory.getLog(RootLogger.class);

public static void main(String[] args) {

if (logger.isDebugEnabled()) {

logger.debug("root logger debuging...");

}

if (logger.isInfoEnabled()) {

logger.info("root logger info...");

}

}

}

package log4j.myLogger;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

public class MyLogger {

// 使用logger名称获取logger

// 申明appender的logger，信息都会被rootLogger包括在内

private static Log logger1 = LogFactory.getLog("myLogger1");

private static Log logger2 = LogFactory.getLog("myLogger2");

public static void main(String[] args) {

// 多个输出流可以通过使用不同的输出级别区分appender

if (logger1.isDebugEnabled()) {

logger1.debug("myLogger1 isDebugEnabled...");

}

if (logger1.isInfoEnabled()) {

logger1.info("myLogger1 isInfoEnabled...");

}

if (logger2.isDebugEnabled()) {

logger2.debug("myLogger2 isDebugEnabled...");

}

if (logger2.isInfoEnabled()) {

logger2.info("myLogger2 isInfoEnabled...");

}

}

}

package log4j.mypk.dbpk;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

public class DbLogger {

// 因为类所在包有对应logger，所以会调用rootLogger以及所有父包对应的logger

private static Log logger = LogFactory.getLog(DbLogger.class);

public static void main(String[] args) {

// TODO Auto-generated method stub

logger.info("DbLogger is infoing...");

}

}

package log4j.mypk;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

public class MyData {

// 没有对应的类名，会调用rootLogger和所有父级包对应的logger

private static Log rootLogger = LogFactory.getLog("rootLogger");

private static Log myLogger1 = LogFactory.getLog("myLogger1");

public static void main(String[] args) {

// TODO Auto-generated method stub

rootLogger.info("rootLogger is infoing...");

myLogger1.info("myLogger1 is infoing...");

}

}

package log4j.mypk;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

public class MyMail {

//没有对应的类名，会调用rootLogger和所有父级包对应的logger

private static Log logger = LogFactory.getLog(MyMail.class);

public static void main(String[] args) {

// TODO Auto-generated method stub

logger.info("MyMail is infoing...");

}

}

## 使用spring的log4j

## 使用AOP的log4j

# 问题与解决

## DailyRollingFileAppender不生成昨天日志

虽然按理说是每个次日生成昨天的日志，但其实在次日必须有日志文件操作才会触发生成。

Log4j设计这种机制的原因在于设计者考虑24点的时候，项目不一定在运行中，所以无法保证每一天都有文件生成。

解决方法：

M1写一个在次日定时任务触发日志操作的任务，保证会产生新日志。

可能遇到的问题？linux下，虽然触发生成昨天日志，但是指向的文件输出流还是对应昨天的日志而不是临时日志，导致今天的日志无法正常记录到临时日志中。（所以我在试验是否可以通过次日打开临时日志的方式保证出发生成昨天的日志）