Log4j学习笔记

# Log4j简单介绍

Log4j是**Apache**的一个开源项目，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**。

Log4j由三个重要的组件构成：日志信息的优先级，日志信息的输出目的地，日志信息的输出格式。日志信息的优先级从高到低有**ERROR、WARN、 INFO、DEBUG**，分别用来**指定这条日志信息的重要程度**；日志信息的输出目的地指定了日志将打印到控制台还是文件中；而输出格式则**控制了日志信息的显示内容**。  
 Log4j由**三个重要的组件**构成：

### 日志信息的优先级（Loggers），

日志信息的优先级从高到低有**ERROR、WARN、 INFO、DEBUG**，分别用来指定这条日志信息的重要程度。（详细见后面介绍）

### 日志信息的输出目的地（Appenders）

日志信息的输出目的地指定了日志将打印到控制台还是文件中。

### 日志信息的输出格式（Layouts）。

输出格式则控制了**日志信息的显示内容**。

# Log4j的配置文件

其实您也可以完全不使用配置文件，而是**在代码中配置Log4j环境**。但是，使用配置文件将使您的应用程序更加灵活。   
**Log4j支持两种配置文件格式，一种是XML格式的文件，一种是properties格式的文件**。

示例：  
log4j.rootLogger=INFO, A1   
log4j.appender.A1=org.apache.log4j.**ConsoleAppender**  
log4j.appender.A1**.layout**=org.apache.log4j.**PatternLayout**  
log4j.appender.A1.**layout.ConversionPattern**=%-4r %-5p [%t] %37c %3x - %m%n

## 配置根Logger

配置根Logger，其语法为：   
**log4j.rootLogger = [ level ] , appenderName, appenderName, …**

### level表示的意义：

**level 是日志记录的优先级**，分为OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL或者您定义的级别。**Log4j建议只使用四个级别，优先级从高到低分别是ERROR、WARN、INFO、DEBUG**。**通过在这里定义的级别，您可以控制到应用程序中相应级别的日志信息的开关。**比如在这里定义了INFO级别，则应用程序中所有DEBUG级别的日志信息将不被打印出来。

### appenderName：

appenderName就是指定**日志信息输出到哪个地方**。您**可以同时指定多个输出目的地**。

## 配置日志信息输出目的地Appender，其语法为：

**log4j.appender.appenderName = package+appender\_class\_name（提供了6种，选择一种即可）**

**log4j.appender.appenderName.option1 = value1**

**…**

**log4j.appender.appenderName.optionN = valueN**

其中，Log4j提供的**appender**有以下几种：

①org.apache.log4j.**ConsoleAppender**（**控制台**），

②org.apache.log4j.FileAppender（**文件**），

③org.apache.log4j.DailyRollingFileAppender（**每天产生一个日志文件**），

④org.apache.log4j.RollingFileAppender（**文件大小到达指定尺寸的时候产生一个新的文件**），

⑤org.apache.log4j.WriterAppender（**将日志信息以流格式发送到任意指定的地方**)

⑥org.apache.log4j.jdbc.JDBCAppender(**将日志写入数据库**)

这6种appendar的选项见后面详细介绍。

## 配置日志信息的布局（格式）（layout属性），其语法为：

log4j.appender.**appenderName.layout** = **package+layout\_class\_name（提供了几种，选择一种即可）**

log4j.appender.appenderName.**layout.option1** = value1

…

log4j.appender.appenderName.**layout.optionN** = valueN

其中，Log4j提供的layout有以下几种：

①org.apache.log4j.HTMLLayout（**以HTML表格形式布局**），

②**org.apache.log4j.PatternLayout（可以灵活地指定布局模式）（使用较多）**，

③org.apache.log4j.SimpleLayout（包含**日志信息的级别和信息字符串**），

④org.apache.log4j.TTCCLayout（包含**日志产生的时间、线程、类别等等信息**）

## 日志的输出格式设置layout.ConversionPattern：

### 指定格式介绍

在配置文件中可以通过log4j.appender.A1.**layout.ConversionPattern**设置日志输出格式。

参数：

#### ① %p: 输出日志信息优先级，即DEBUG，INFO，WARN，ERROR，FATAL,

#### ②%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：2002年10月18日 22：10：28，921

显示日志记录时间，{<日期格式>}使用ISO8601定义的日期格式。

%d{ISO8601} 格式为 2005-10-12 22:23:30,117

#### ③%r: 输出自应用启动到输出该log信息耗费的毫秒数；

#### ④%c: 输出日志信息所属的类目，通常就是所在类的全名；

列出logger名字空间的全称，如果加上**{<层数>}**表示列出从最内层算起的指定层数的名字空间。

#### ⑤%C：列出调用logger的类的全名（包含包路径）

#### ⑥%l: 输出日志事件的发生位置，相当于%C.%M(%F:%L)的组合,包括类目名、发生的线程，以及在代码中的行数。举例：Testlog4.main(TestLog4.Java:10)

#### ⑦%x: 输出和当前线程相关联的NDC(嵌套诊断环境),尤其用到像Java servlets这样的多客户多线程的应用中。

#### ⑧%%: 输出一个”%”字符

#### ⑨%F: 输出日志消息产生时所在的文件名称

#### ⑩%L: 输出代码中的行号

#### %m: 输出代码中指定的消息,产生的日志具体信息

#### %n: 输出一个回车换行符，Windows平台为”\r\n”，Unix平台为”\n”输出日志信息换行。

#### %t: 输出产生该日志事件的线程名。

注：更多的及示例见本文最后表格。

### 可以在%与模式字符之间加上修饰符来控制其最小宽度、最大宽度、和文本的对齐方式。如：

#### 1)%20c：指定输出category的名称，最小的宽度是20，如果category的名称小于20的话，默认的情况下右对齐。

#### 2)%-20c:指定输出category的名称，最小的宽度是20，如果category的名称小于20的话，”-”号指定左对齐。

#### 3)%.30c:指定输出category的名称，最大的宽度是30，如果category的名称大于30的话，就会将左边多出的字符截掉，但小于30的话也不会有空格。

#### 4)%20.30c:如果category的名称小于20就补空格，并且右对齐，如果其名称长于30字符，就从左边交远销出的字符截掉。

## 配置输出的编码集

#### log4j.appender.appendarName.Encoding=UTF-8

其中appendarName就是具体的appendarName名字。

# 如何在不同的模块中输出不同的日志

用户基础信息模块路径为：com.test.user

它下面有个类：com.test.user.service.impl.UserInfo

private Log log = LogFactory.getLog(UserInfo.class);

## 方法1：在log4j.properties中加入:

log4j.logger.com.test.user=info,userLog,stdout

log4j.appender.userLog=org.apache.log4j.FileAppender

log4j.appender.userLog.File=../logs/userinfo.log

log4j.appender.userLog.Append=true

log4j.appender.userLog.Threshold=info

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

log4j.appender.userLog.layout.ConversionPattern==%d %p [%c] - %m%n

注：也就是让com.test.user模块下所有的logger使用log4j.appender.userLog所做的配置。

## 方法2：自定义“别名”

private Log log = **LogFactory.getLog("userInfoLog");**

然后在log4j.properties中加入:

log4j.logger.userInfoLog=info,userLog,stdout

log4j.appender.userLog=org.apache.log4j.FileAppender

log4j.appender.userLog.File=../logs/userinfo.log

log4j.appender.userLog.Append=true

log4j.appender.userLog.Threshold=info

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

log4j.appender.userLog.layout.ConversionPattern==%d %p [%c] - %m%n

注：也就是在用logger时给它一个自定义的名字(如这里的"userInfoLog")，然后在log4j.properties中做出相应配置即可。，在这种模式下，即使在同一个类中也能定义多个不同输出的log.

在类中调用代码如下：  
private Log loggerError = LogFactory.getLog("userErrorLog");  
private Log loggerInfo = LogFactory.getLog("userInfoLog");

自定义的日志默认是同时输出到log4j.rootLogger所配置的日志中的，如何能只让它们输出到自己指定的日志中呢？

**log4j.additivity.userInfoLog = false**  
它用来设置是否同时输出到log4j.rootLogger所配置的日志中，设为false就不会输出到其它地方啦！注意这里的"userInfoLog"是你在程序中给logger起的那个自定义的名字！  
如果你说，我只是不想同时输出这个日志到log4j.rootLogger所配置的logfile中，stdout里我还想同时输出呢！如：  
log4j.logger.userInfoLog=DEBUG, userLog, **stdout**

# 加载log4j.properties文件

## Log4j加载配置文件时：

### 如果是Tomcat，则会默认寻找$TOMCAT\_HOME/lib/log4j.xml或者log4j.properties，如果指定了Log4j的位置，也会会从指定的位置处寻找Log4j的配置文件。

#### 如果是自己的一个Web应用，则默认会寻找该应用的/WEB-INF/classes(或者lib)/log4j.xml或log4j.properties。也可以从配置Log4j的位置处加载log4j.xml或log4.properties。

**对于WEB-INF下是classes优先还是lib优先 你可以自己测试一下。**

**因此，实际开发中，只需要把log4j.xml或log4j.properties配置文件放在src目录下，生成war包或编译运行时，默认将此配置文件放到/WEB-INF/classes下。放到src或src/main/java下，都是一样的，因为编译的时候类都变成了class文件，都在classes目录下。确保最后log4j.xml或log4j.properties在classes目录下即可自动加载。**

## spring方式加载，配置与web.xml中：[spring](http://lib.csdn.net/base/javaee)**方式加载，配置与web.xml中：**

**Spring加载log4j.properties，它提供了一个Log4jConfigListener，**本身就能通过web.xml配置从指定位置加载log4j配置文件和log4j的输出路径，**要注意的是Log4jConfigListener必须要在Spring的Listener之前。**

web.xml

**<!-- 设置由Spring载入的Log4j配置文件位置 -->**

**<context-param>**

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

**<param-value>WEB-INF/classes/log4j.properties</param-value>**

**</context-param>**

**<!-- Spring刷新Log4j配置文件变动的间隔,单位为毫秒 -->**

**<context-param>**

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

**<param-value>10000</param-value>**

**</context-param>**

**<listener>**

**<listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>**

**</listener>**

## **可以通过资源类对资源文件进行加载，与使用为一体**

public class Logger4JTest {

public static void main(String[] args) {

**PropertyConfigurator.configure(" D:/log/log4j.properties ");**

**Logger logger = Logger.getLogger(Logger4JTest.class);**

logger.debug(" debug ");

logger.error(" error ");

}

}

# 在程序中Log4j的具体使用

在程序中使用Log4j之前，首先要将**commons-logging.jar和logging-log4j-1.2.9.jar**导入**到classpath**中，并将**log4j.properties放于src根目录中**。接下来就可以使用了。这是默认位置，不需要再手动配置了。

## 得到记录器

使用Log4j，第一步就是**获取日志记录器**，**这个记录器将负责控制日志信息**。其语法为：   
public static Logger getLogger( String name)，   
 **通过指定的名字获得记录器，如果必要的话，则为这个名字创建一个新的记录器**。Name一般取本类的名字，比如：   
static Logger logger = Logger.getLogger ( ServerWithLog4j.class.getName () ) ;

应该是LoggerFactory获取Logger：

private static final Logger logger = **LoggerFactory.getLogger**(MyController.class);

## 插入记录信息（格式化日志信息） 当上两个必要步骤执行完毕，您就可以轻松地使用不同优先级别的日志记录语句插入到您想记录日志的任何地方，其语法如下：  logger.debug ( Object message ) ;  logger.info ( Object message ) ;  logger.warn ( Object message ) ;  logger.error ( Object message ) ;

# 六种appendar的详细介绍

## **ConsoleAppender选项**

Threshold=WARN:**指定日志消息的输出最低层次**。

ImmediateFlush=true:默认值是true,**意谓着所有的消息都会被立即输出**。

Target=System.err：**默认情况下是：System.out,指定输出控制台**

## FileAppender 选项

Threshold=WARN:指定日志消息的输出最低层次。

ImmediateFlush=true:默认值是true,意谓着所有的消息都会被立即输出。

File=mylog.txt:**指定消息输出到mylog.txt文件**。

Append=false:**默认值是true,即将消息增加到指定文件中，false指将消息覆盖指定的文件内容**。

## **DailyRollingFileAppender 选项**

Threshold=WARN:指定日志消息的输出最低层次。

ImmediateFlush=true:默认值是true,意谓着所有的消息都会被立即输出。

**File=mylog.txt:指定消息输出到mylog.txt文件。（每天先放到 该文件中过了一（加入按天），再拷贝到DatePattern指定的文件中）**

Append=false**:默认值是true,即将消息增加到指定文件中，false指将消息覆盖指定的文件内容。**

DatePattern=**’.'yyyy-ww**:**每周滚动一次文件，即每周产生一个新的文件。当然也可以指定按月、周、天、时和分**。即对应的格式如下：

1)’.'yyyy-MM: 每月

2)’.'yyyy-ww: 每周

3)’.**'yyyy-MM-dd**: 每天

4)**’.'yyyy-MM-dd-a: 每天两次**

5)’.'yyyy-MM-dd-HH: 每小时

6)’.'yyyy-MM-dd-HH-mm: 每分钟

## RollingFileAppender 选项

Threshold=WARN:**指定日志消息的输出最低层次**。

ImmediateFlush=true:默认值是true,意谓着所有的消息都会被立即输出。

File=mylog.txt:指定消息输出到mylog.txt文件。

Append=false:默认值是true,即将消息增加到指定文件中，false指将消息覆盖指定的文件内容。

**MaxFileSize=100KB: 后缀可以是KB, MB 或者是 GB. 在日志文件到达该大小时，将会自动滚动，即将原来的内容移到mylog.log.1文件。**

**MaxBackupIndex=2:指定可以产生的滚动文件的最大数**。

## **JDBCApperder选项**

URL=jdbc:[**MySQL**](http://lib.csdn.net/base/mysql)://localhost:3306/test：**指定日志写入的数据库链接**  
**driver=com.mysql.jdbc.Driver：指定数据库驱动**  
user=root：指定数据库的用户名  
password=123：指定数据库的登录密码  
sql=insert into tb\_log (message) values('=[%-5p] %d(%r) --> [%t] %l: %m %x %n')：**指定写入数据库的执行语句。**

# Log4j的日志级别

每个**Logger**都被了一个**日志级别（log level）**，用来控制日志信息的输出。日志级别从高到低分为：

A：off 最高等级，用于关闭所有日志记录。

B：fatal 指出每个严重的错误事件将会导致应用程序的退出。

**C：error 指出虽然发生错误事件，但仍然不影响系统的继续运行。**

**D：warm 表明会出现潜在的错误情形。**

**E：info 一般和在粗粒度级别上，强调应用程序的运行全程。**

**F：debug 一般用于细粒度级别上，对调试应用程序非常有帮助。**

G：all 最低等级，用于打开所有日志记录。

上面这些级别是定义在**org.apache.log4j.Level**类中。Log4j只建议使用4个级别，优先级从高到低分别是**error,warn,info和debug**。通过使用日志级别，可以控制应用程序中相应级别日志信息的输出。例如，如果使用了info级别，则应用程序中所有低于info级别的日志信息(如debug)将不会被打印出来。

# Log4j输出格式控制--log4j的PatternLayout参数含义

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **说明** | **例子** | |
| %c | 列出logger名字空间的全称，如果加上{<层数>}表示列出从最内层算起的指定层数的名字空间 | log4j配置文件参数举例 | 输出显示媒介 |
| 假设当前logger名字空间是"a.b.c" | |
| %c | a.b.c |
| **%c{2}** | **b.c** |
| %20c | （**若名字空间长度小于20，则左边用空格填充**） |
| %-20c | （**若名字空间长度小于20，则右边用空格填充**） |
| %.30c | （**若名字空间长度超过30，截去多余字符**） |
| %20.30c | （**若名字空间长度小于20，则左边用空格填充；若名字空间长度超过30，截去多余字符**） |
| %-20.30c | （**若名字空间长度小于20，则右边用空格填充；若名字空间长度超过30，截去多余字符**） |
| %C | 列出调用logger的类的全名（包含包路径） | 假设当前类是"org.apache.xyz.SomeClass" | |
| %C | org.apache.xyz.SomeClass |
| %C{1} | SomeClass |
| %d | 显示日志记录时间，{<日期格式>}使用ISO8601定义的日期格式 | **%d{yyyy/MM/dd HH:mm:ss,SSS}** | 2005/10/12 22:23:30,117 |
| %d{ABSOLUTE} | 22:23:30,117 |
| %d{DATE} | 12 Oct 2005 22:23:30,117 |
| **%d{ISO8601}** | **2005-10-12 22:23:30,117** |
| %F | 显示调用logger的源文件名 | %F | MyClass.java |
| %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 | %l | MyClass.main(MyClass.java:129) |
| %L | 显示调用logger的代码行 | %L | 129 |
| %m | 显示输出消息 | %m | This is a message for debug. |
| %M | 显示调用logger的方法名 | %M | main |
| %n | 当前平台下的换行符 | %n | Windows平台下表示rn UNIX平台下表示n |
| %p | 显示该条日志的优先级 | %p | INFO |
| %r | 显示从程序启动时到记录该条日志时已经经过的毫秒数 | %r | 1215 |
| %t | 输出产生该日志事件的线程名 | %t | MyClass |
| %x | 按NDC（Nested Diagnostic Context，线程堆栈）顺序输出日志 | 假设某程序调用顺序是MyApp调用com.foo.Bar | |
| %c %x - %m%n | MyApp - Call com.foo.Bar. com.foo.Bar - Log in Bar MyApp - Return to MyApp. |
| %X | 按MDC（Mapped Diagnostic Context，线程映射表）输出日志。通常用于多个客户端连接同一台服务器，方便服务器区分是那个客户端访问留下来的日志。 | %X{5} | （记录代号为5的客户端的日志） |
| %% | 显示一个百分号 | %% | % |