# 概念

日志是应用软件中不可缺少的部分，Apache的开源项目[log4j](http://www.codeceo.com/article/log4j-usage.html)是一个功能强大的日志组件,提供方便的日志记录。

log4j由三个重要的组件构成：Loggers(记录器)，Appenders (输出源)和Layouts(布局)。

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基本语法（重点）

## 定义配置文件

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Java属性文件（键=值），下面我们介绍使用Java属性文件做为配置文件的方法。

### （1）配置根Logger

#### 语法

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

#### 说明

其中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就是指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。

#### 示例

log4j.rootLogger = INFO , D

### （2）配置日志信息的输出目的地Appender

#### 语法

log4j.appender.appenderName = className

#### 说明

其中Log4j提供的appender有以下className：

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）

##### ConsoleAppender选项

Threshold=WARN：指定日志信息的最低输出级别，默认为DEBUG。

ImmediateFlush=true：表示所有消息都会被立即输出，设为false则不输出，默认值是true。

Target=System.err：默认值是System.out

##### FileAppender选项

Threshold=WARN：指定日志信息的最低输出级别，默认为DEBUG。

ImmediateFlush=true：表示所有消息都会被立即输出，设为false则不输出，默认值是true。

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

File=D:/logs/logging.log4j：指定消息输出到logging.log4j文件中

##### DailyRollingFileAppender选项

**Threshold**=WARN：指定日志信息的最低输出级别，默认为DEBUG。

ImmediateFlush=true：表示所有消息都会被立即输出，设为false则不输出，默认值是true。

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

**File**=D:/logs/logging.log4j：指定当前消息输出到logging.log4j文件中。

DatePattern='.'yyyy-MM：每月滚动一次日志文件，即每月产生一个新的日志文件。当前月的日志文件名为logging.log4j，前一个月的日志文件名为logging.log4j.yyyy-MM

##### RollingFileAppender选项

Threshold=WARN：指定日志信息的最低输出级别，默认为DEBUG。

ImmediateFlush=true：表示所有消息都会被立即输出，设为false则不输出，默认值是true。

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

File=D:/logs/logging.log4j：指定消息输出到logging.log4j文件中。

MaxFileSize=100KB：后缀可以是KB, MB 或者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logging.log4j.1文件中。

MaxBackupIndex=2：指定可以产生的滚动文件的最大数，例如，设为2则可以产生logging.log4j.1，logging.log4j.2两个滚动文件和一个logging.log4j文件。

#### 示例

log4j.appender.D = org.apache.log4j.DailyRollingFileAppender

### （3）配置日志信息的格式（布局）

#### 语法

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

#### 说明

其中，Log4j提供的layout有以下几种：

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）

##### HTMLLayout选项

LocationInfo=true：输出java文件名称和行号，默认值是false。

Title=My Logging： 默认值是Log4J Log Messages。

##### PatternLayout选项

ConversionPattern=%m%n：设定以怎样的格式显示消息。

#### 示例

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

### （4）配置打印参数格式

#### 说明

Log4J采用类似C语言中的printf函数的打印格式格式化日志信息，打印参数如下：

%m 输出代码中指定的消息

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

%p 输出优先级，即DEBUG，INFO，WARN，ERROR，FATAL

%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

%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 MMM dd HH:mm:ss,SSS}，输出类似：2002年10月18日 22：10：28，921

%l 输出日志事件的发生位置，包括类名、发生的线程，以及在代码中的行数。举例：Testlog4.main(TestLog4.java:10)

%L: 输出代码中的行号

#### 示例

log4j.appender.D.layout.ConversionPattern = [%p] [%-d{yyyy-MM-dd HH:mm:ss}] %C.%M(%L) | %m%n

## 编写代码示例

### （1）获取日志记录器

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 () )

### （2）读取配置文件

当获得了日志记录器之后，第二步将配置Log4j环境，其语法为：

BasicConfigurator.configure ()： 自动快速地使用缺省Log4j环境

PropertyConfigurator.configure ( String configFilename) ：读取Java的属性文件。

DOMConfigurator.configure ( String filename ) ：读取XML形式的配置文件。

说明：这步不是必须的，可以省略。

### （3）插入记录信息

当以上两个步骤执行完毕，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 ) ;

### （4）日志级别

每个Logger都有一个日志级别（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。

上面的这些级别是定义在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。

# 常用的类

## Logger

### 基本概念

这是log4j包中的中心类。除配置外，大多数日志记录操作都是通过此类完成的。

### 方法

#### static public Logger getLogger(String name)

检索根据name参数的值命名的记录器。如果已存在指定的记录器，则将返回现有实例。 否则，将创建一个新实例。默认情况下，记录器没有设置级别，但是从具有设置级别的最近祖先继承它，这是log4j的核心功能之一。

参数：name 查询的记录器的名称

#### static public Logger getLogger(Class clazz)

参数：clazz的名称将用作要检索的记录器的名称。

#### public void error(Object message, Throwable t)

参数：

message 要记录的消息对象

t记录的异常，包括其堆栈跟踪。

说明：使用ERROR级别记录消息对象，包括作为参数传递的{@link Throwable} t的堆栈跟踪

#### public void error(Object message)

记录error级别的信息对象

参数：

message 要记录的消息对象

说明：记录error级别的信息对象，此方法首先通过将此类别的级别与ERROR级别进行比较来检查此类别是否已启用ERROR。如果此类别启用了ERROR，则它会通过调用相应的ObjectRenderer将作为参数传递的消息对象转换为字符串。它继续调用此类别中的所有已注册的appender，并且在层次结构中也更高，具体取决于additivity标志的值。警告请注意，将Throwable传递给此方法将打印Throwable的名称但不显示堆栈跟踪。要打印堆栈跟踪，请使用error（Object，Throwable）这个方法。

# log4j配置文件示例（重点）

（1）新建一个maven项目

（2）在src/main/resources目录下面新建log4j.properties文件，然后在文件中编写内容。

## 设置日志级别

log4j.rootLogger = **debug,stdout,D,E**

## 输出日志信息到控制台

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

log4j.appender.stdout.Target = System.out

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

log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%n%m%n

## 输出日志信息到文件

log4j.appender.logFile=org.apache.log4j.FileAppender

log4j.appender.logFile.Threshold=DEBUG

log4j.appender.logFile.ImmediateFlush=true

log4j.appender.logFile.Append=true

log4j.appender.logFile.File=D:/logs/log.log4j

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

log4j.appender.logFile.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n

## 输出日志信息到回滚文件

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

log4j.appender.rollingFile.Threshold=DEBUG

log4j.appender.rollingFile.ImmediateFlush=true

log4j.appender.rollingFile.Append=true

log4j.appender.rollingFile.File=D:/logs/log.log4j

log4j.appender.rollingFile.MaxFileSize=200KB

log4j.appender.rollingFile.MaxBackupIndex=50

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

log4j.appender.rollingFile.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n

## 输出日志信息到定期回滚日志文件

log4j.appender.D = org.apache.log4j.DailyRollingFileAppender

log4j.appender.D.File =log/info.lof

log4j.appender.D.Append = true

log4j.appender.D.Threshold = DEBUG

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

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

## 应用于Socket

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

log4j.appender.socket.RemoteHost=localhost

log4j.appender.socket.Port=5001

log4j.appender.socket.LocationInfo=true

## 发送日志到指定邮件

log4j.appender.mail=org.apache.log4j.net.SMTPAppender

log4j.appender.mail.Threshold=FATAL

log4j.appender.mail.BufferSize=10

log4j.appender.mail.From = xxx@mail.com

log4j.appender.mail.SMTPHost=mail.com

log4j.appender.mail.Subject=Log4J Message

log4j.appender.mail.To= xxx@mail.com

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

log4j.appender.mail.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n

## 应用于数据库

log4j.appender.database=org.apache.log4j.jdbc.JDBCAppender

log4j.appender.database.URL=jdbc:mysql://localhost:3306/test

log4j.appender.database.driver=com.mysql.jdbc.Driver

log4j.appender.database.user=root

log4j.appender.database.password=

log4j.appender.database.sql=INSERT INTO LOG4J (Message) VALUES('=[%-5p] %d(%r) --> [%t] %l: %m %x %n')

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

log4j.appender.database.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n

## 自定义Appender

log4j.appender.im = net.cybercorlin.util.logger.appender.IMAppender

log4j.appender.im.host = mail.cybercorlin.net

log4j.appender.im.username = username

log4j.appender.im.password = password

log4j.appender.im.recipient = corlin@cybercorlin.net

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

log4j.appender.im.layout.ConversionPattern=[%-5p] %d(%r) --> [%t] %l: %m %x %n

# Web项目中使用Log4j的方法

在J2EE应用中使用Log4j，必须先在启动服务时加载Log4j的配置文件进行初始化，可以在web.xml中进行配置。

（1）web应用的log4j使用基本上都采用：新建一个servlet，这个servlet在init函数中为log4j执行配置，一般就是读入配置文件，所以需要在web.xml中为这个servlet配置，同时设定load-on-startup为1。

（2）这个servlet配置log4j就是读出配置文件，然后调用configure函数。这里有两个问题：一、需要知道文件在哪里；二、需要正确的文件类型。

（3）配置文件位置在web.xml中配置一个param元素即可，路径一般是相对于web的root目录。

（4）文件类型一般有两种，一个是Java的property文件，另一种是xml文件。

# 将Java异常的完整堆栈内容打印到文件中

异常打印建议：logger.error("业务名称",e);

其中，logger是org.apache.log4j.Logger的对象，e是完整的异常信息。

# 把需要的内容从原有日志中分离，形成单独的日志文件

先看一个常见的log4j.properties文件，它是在控制台和myweb.log文件中记录日志：

log4j.rootLogger=DEBUG, stdout, logfile

log4j.category.org.springframework=ERROR

log4j.category.org.apache=INFO

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=${myweb.root}/WEB-INF/log/myweb.log

log4j.appender.logfile.MaxFileSize=512KB

log4j.appender.logfile.MaxBackupIndex=5

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

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

如果想对不同的类输出不同的文件(以cn.com.Test为例)，先要在Test.java中定义:

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

然后在log4j.properties中加入:

log4j.logger.cn.com.Test= DEBUG, **test**

log4j.appender.**test**=org.apache.log4j.FileAppender

log4j.appender.test.File=${myweb.root}/WEB-INF/log/**test.log**

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

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

也就是让cn.com.Test中的logger使用log4j.appender.test所做的配置

但是，如果在同一类中需要输出多个日志文件呢？其实道理是一样的，先在Test.java中定义:

private static Log logger1 = LogFactory.getLog("myTest1");

private static Log logger2 = LogFactory.getLog("myTest2");

然后在log4j.properties中加入:

log4j.logger.myTest1= DEBUG, test1

log4j.appender.test1=org.apache.log4j.FileAppender

log4j.appender.test1.File=${myweb.root}/WEB-INF/log/**test1.log**

log4j.appender.test1.layout=org.apache.log4j.PatternLayout

log4j.appender.test1.layout.ConversionPattern=%d %p [%c] - %m%n

log4j.logger.myTest2= DEBUG, test2

log4j.appender.test2=org.apache.log4j.FileAppender

log4j.appender.test2.File=${myweb.root}/WEB-INF/log/**test2.log**

log4j.appender.test2.layout=org.apache.log4j.PatternLayout

log4j.appender.test2.layout.ConversionPattern=%d %p [%c] - %m%n

也就是在用logger时给它一个自定义的名字(如这里的"myTest1")，然后在log4j.properties中做出相应配置即可。别忘了不同日志要使用不同的logger(如输出到**test1.log**的要用**logger1**.info("abc"))

还有一个问题，就是这些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？**别急，这里有个开关：

log4j.**additivity**.myTest1 = 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"myTest1"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！那也好办，把你的log4j.logger.myTest1 = DEBUG, test1改为下式就OK啦！

log4j.logger.myTest1=DEBUG, test1

示例

下面是文件上传时记录文件类型的log日志，并输出到指定文件的配置



Java端控制代码

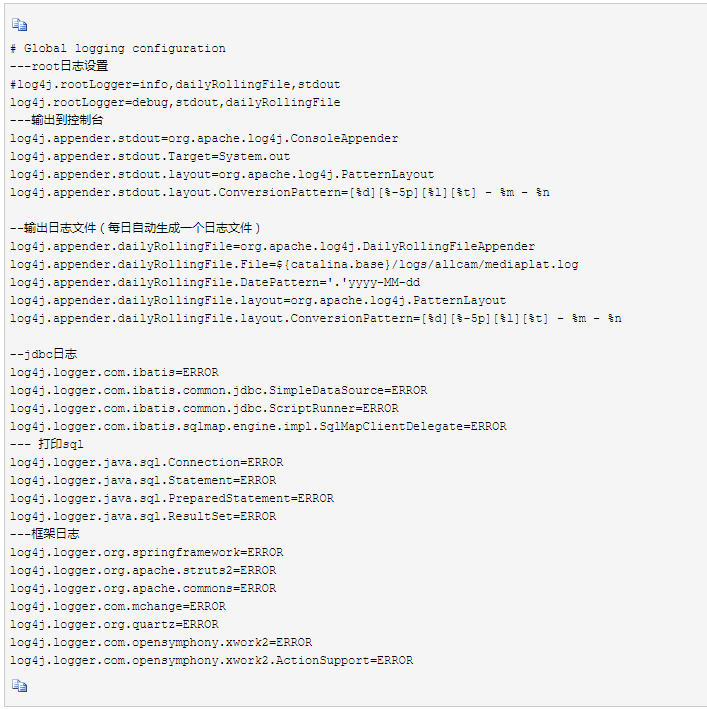


# maven项目中配置log4j

## 1、在maven中引入依赖



## 2、在resource文件夹下添加log4j.properties文件



## 3、在web.xml中进行日志监听



说明：这步可以省略不要

## 4、测试

