****

**SSM前言**

**课程讲义**

**北京动力节点教育培训公司**

**主讲：Reyco·郭**

**SSM前言**

**[www.bjpowernode.com](http://www.bjpowernode.com)**

**SSM前言讲义**

# Log4j

一个完整的软件，日志是必不可少的。程序从开发、测试、维护、运行等环节，都需要向控制台或文件等位置输出大量信息。这些信息的输出，在很多时候是使用System.out.println()无法完成的。

日志信息根据用途与记录内容的不同，分为调试日志、运行日志、异常日志等。

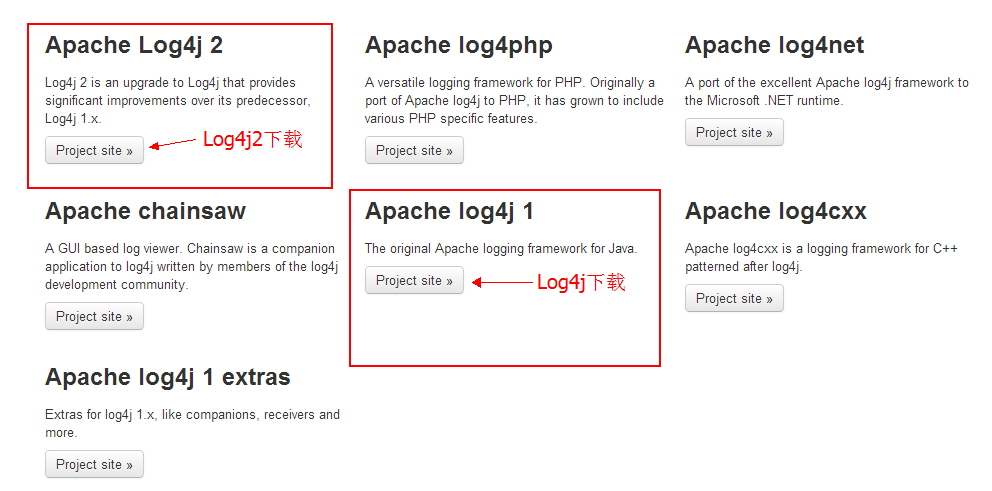
用于日志记录的技术很多，如jdk的logger技术，apache的log4j、log4j2技术等。

Log4j的全称为Log for java，即，专门用于java语言的日志记录工具。

## Log4j基础

### Log4j的下载

Log4j下载地址：http://logging.apache.org/

****

#### Log4j版本下载

### 日志级别

为了方便对于日志信息的输出显示，对日志内容进行了分级管理。日志级别由高到低 ，共分6个级别：fatal(致命的)、error、warn、info、debug、trace(堆栈)。

为什么要对日志进行分级呢？

无论是将日志输出到控制台，还是文件，其输出都会降低程序的运行效率。但由于调试、运行维护的需要，客户的要求等原因，需要进行必要的日志输出。这时就必须要在代码中加入日志输出语句。

这些输出语句若在程序运行时全部执行，则势必会降低运行效率。例如，使用System.out.println()将信息输出到控制台，则所有的该输出语句均将执行。会大大降低程序的执行效率。而要使其不输出，唯一的办法就是将这些输出语句逐个全部删除。这是个费时费力的过程。

将日志信息进行分级管理，便可方便的控制信息输出内容及输出位置：哪些信息需要输出，哪些信息不需要输出，只需在一个日志输出控制文件中稍加修改即可。而代码中的输出语句不用做任何修改。

从这个角度来说，代码中的日志编写，其实就是写大量的输出语句。只不过，这些输出语句比较特殊，它们具有级别，在程序运行期间不一定被执行。它们的执行是由另一个控制文件控制。

### 日志输出控制文件

Log4j的日志输出控制文件，主要由三个部分构成：

（1）日志信息的输出位置：控制日志信息将要输出的位置，是控制台还是文件等。

（2）日志信息的输出格式：控制日志信息的显示格式，即以怎样的字符串形式显示。

（3）日志信息的输出级别：控制日志信息的显示内容，即显示哪些级别的日志信息。

有了日志输出控制文件，代码中只要设置好日志信息内容及其级别即可，通过控制文件便可控制这些日志信息的输出了。

## Log4j技术

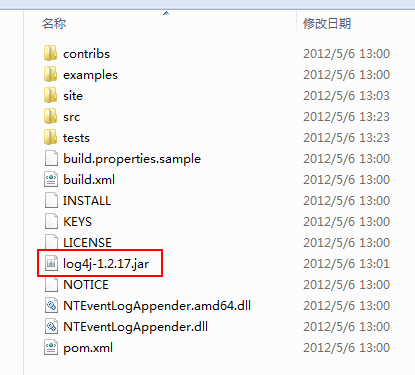
### 在程序中的日志实现步骤

项目：log4j

若要在自己的程序中写入日志语句，则可按照以下步骤进行：

#### 导入Jar包

在项目中导入log4j需要的jar包。将Log4j框架的zip文件进行解压，在其根目录下就有其Jar包。

****

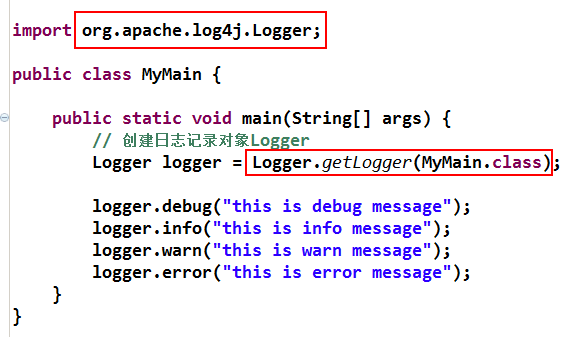
#### 放入日志输出控制文件

将属性文件log4j.properties直接放到项目的src下

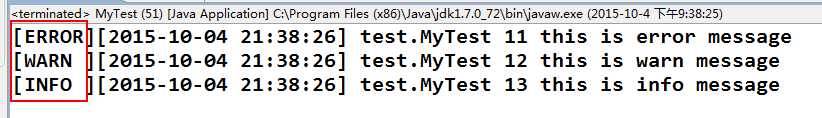
#### 代码中实现日志记录

在要输出日志的类中创建日志对象Logger，并通过Logger的方法在代码中加入日志输出语句。在Java代码中进行日志输出，需要用到Logger类的静态方法getLogger()。

注意，Logger为org.apache.log4j包中的类。

****

将来这些日志输出语句，会根据log4j.properties文件中日志级别的设置进行输出，会输出到指定位置。其输出结果是：输出指定级别及其更高级别的信息。如指定info级别，则会输出fatal、error、warn、info级别的信息。就本例而言，会执行以下三句，而不会执行debug()方法。

****

### 日志输出控制文件分析

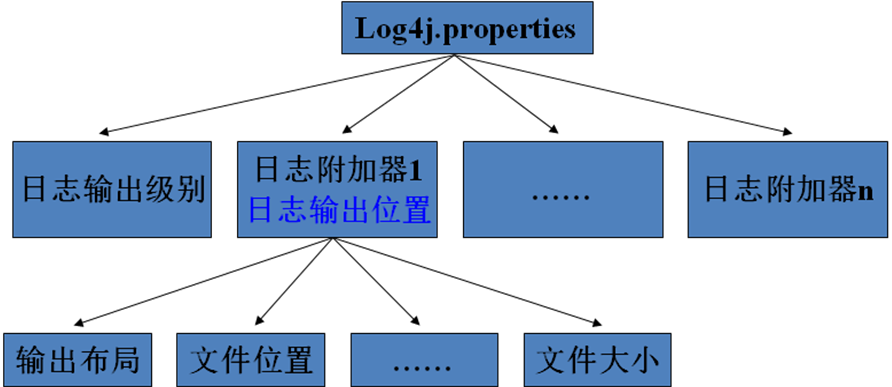
日志属性文件log4j.properties是专门用于控制日志输出的。其主要进行三方面控制：

* 输出位置：控制日志将要输出的位置，是控制台还是文件等。
* 输出布局：控制日志信息的显示形式
* 输出级别：控制要输出的日志级别。

日志属性文件由两个对象组成：日志附加器与根日志。

根日志，即为Java代码中的日志记录器，其主要由两个属性构成：日志输出级别与日志附加器。

日志附加器，则由日志输出位置定义，由其它很多属性进行修饰，如输出布局、文件位置、文件大小等。

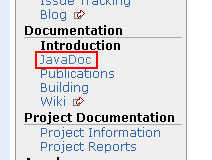
****

#### 定义日志附加器appender

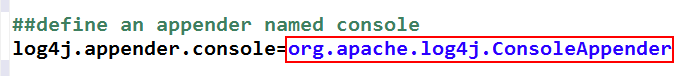
所谓日志附加器，就是为日志记录器附加上很多其它设置信息。附加器的本质是一个接口，其定义语法为：log4j.appender.appenderName = 输出位置

appenderName为自定义名称。

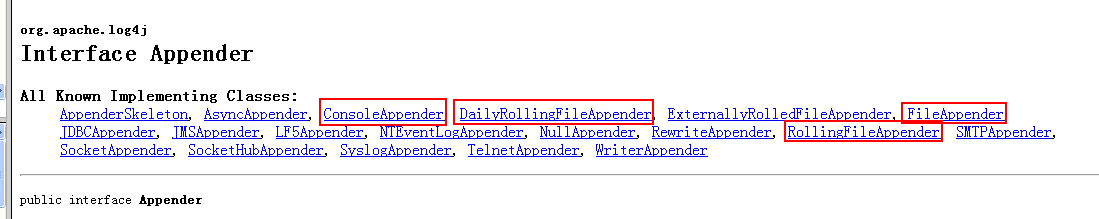
输出位置为log4j指定的类型，是定义好的一些appender接口的实现类。查看log4j框架解压目录下的站点目录site的index.jsp中的JavaDoc，可看到log4j的API。

****

例如定义了一个名称为console的控制台附加器：

****

常用的附加器实现类如下：

****

* org.apache.log4j.ConsoleAppender：日志输出到控制台
* org.apache.log4j.FileAppender：日志输出到文件
* org.apache.log4j.RollingFileAppender：当日志文件大小到达指定尺寸的时候将产生一个新的日志文件
* org.apache.log4j.DailyRollingFileAppender：每天产生一个日志文件

#### 修饰日志附加器

所谓修饰日志附加器，就是为定义好的附加器添加一些属性，以控制到指定位置的输出。不同的附加器，其修饰属性不同。

* 控制台附加器：

Target：控制输出到控制台的使用目标。其值为System.out 或 Sytem.err。它们的区别是，System.out是以黑色字体显示到控制台，而System.err则是以红色字体显示。

* 文件附加器：

File：日志要输出的文件位置及文件名称。

* 滚动文件附加器：

MaxFileSize：用于指定日志文件的最大值。若文件超过指定值，将自动产生另一个日志文件。

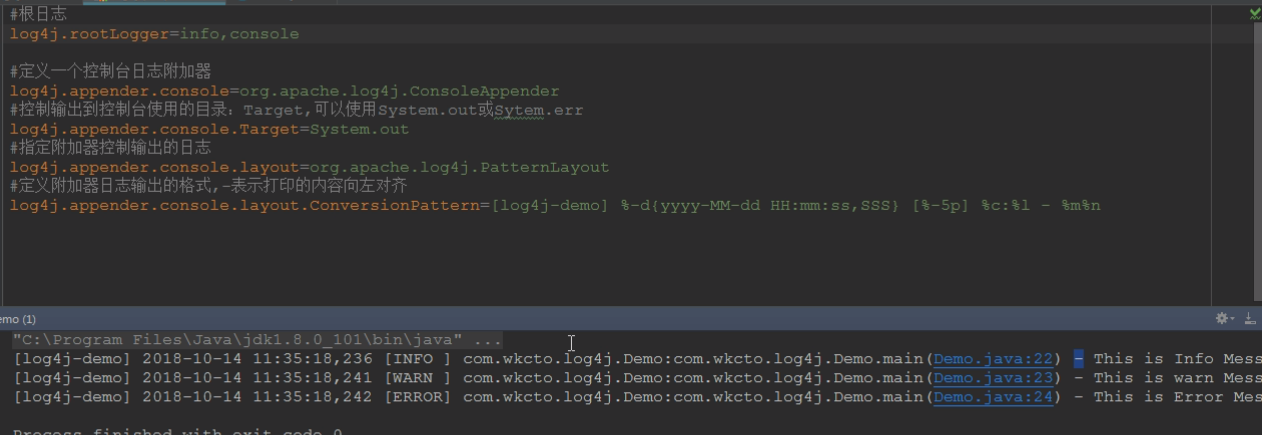
Log4j常用布局类型：

* org.apache.log4j.HTMLLayout：网页布局，以HTML表格形式布局
* org.apache.log4j.SimpleLayout：简单布局，包含日志信息的级别和信息字符串
* org.apache.log4j.PatternLayout：匹配器布局，可以灵活地指定布局模式。其主要是通过设置PatternLayout的ConversionPattern属性值来控制具体输出格式的。ConversionPattern的值中有很多控制字符，这些字符的意义如下表所示：

ConversionPattern取值说明

（本表格来自互联网）

|  |  |  |  |
| --- | --- | --- | --- |
| **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{2} | xyz.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 | %% | % |



#### 配置根Logger

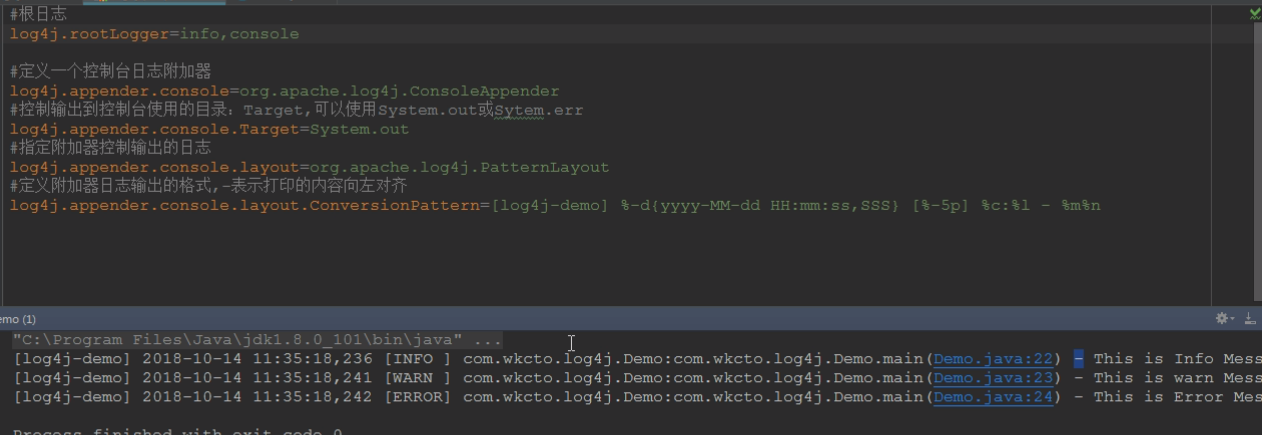
配置rootLogger，以便于代码加载来控制日志的输出。其语法为：

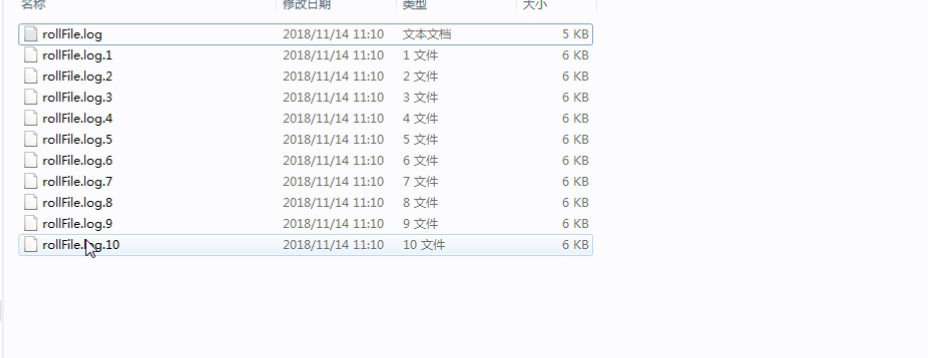
log4j.rootLogger = [ level ] , appenderName, …

其中，level 是日志记录的优先级，分为OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL。Log4j建议只使用四个级别，优先级从高到低分别是ERROR、WARN、INFO、DEBUG。OFF为关闭日志功能。

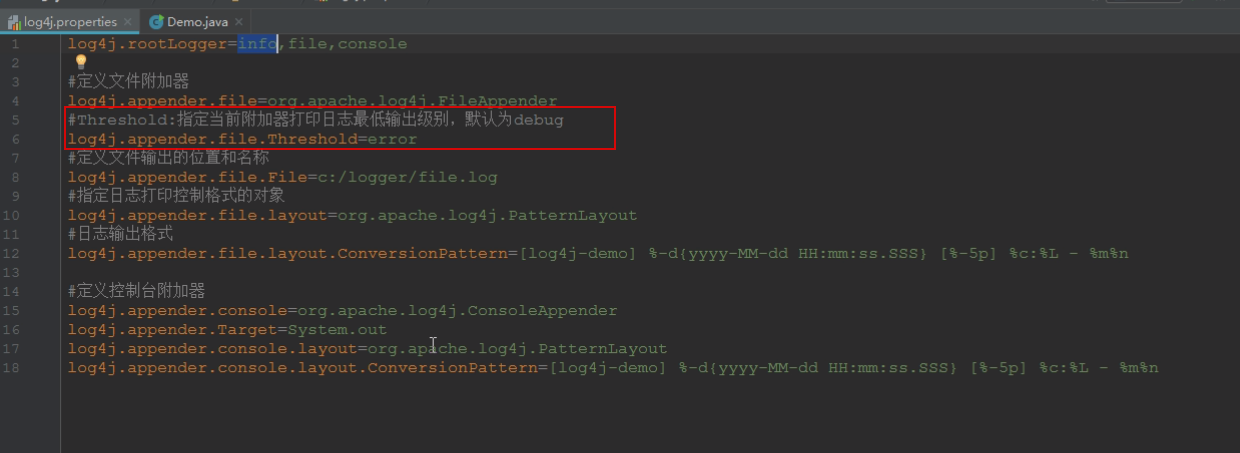
低级别的可以显示高级别的，但高级别的不能显示低级别的。所以，级别越高，将来显示的信息就越少。

#### 滚动文件附加器

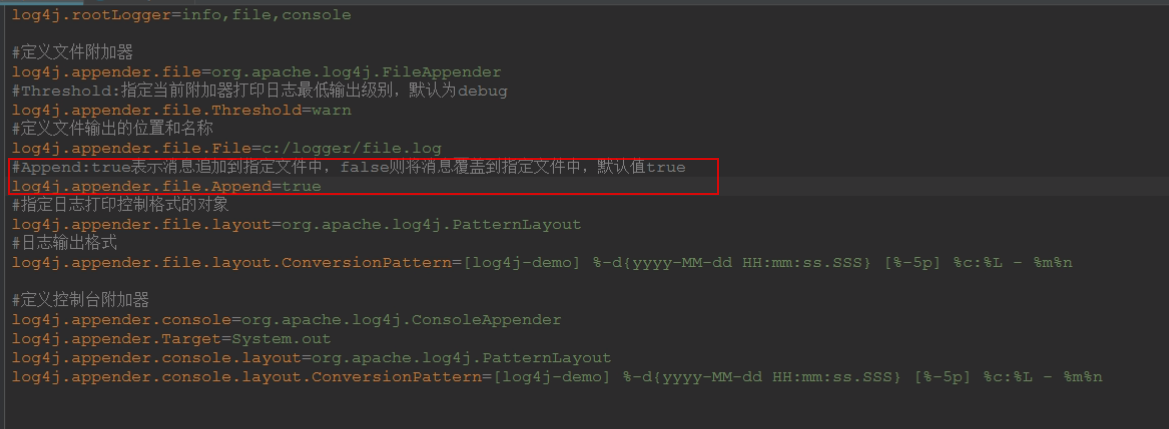




#### Threshold

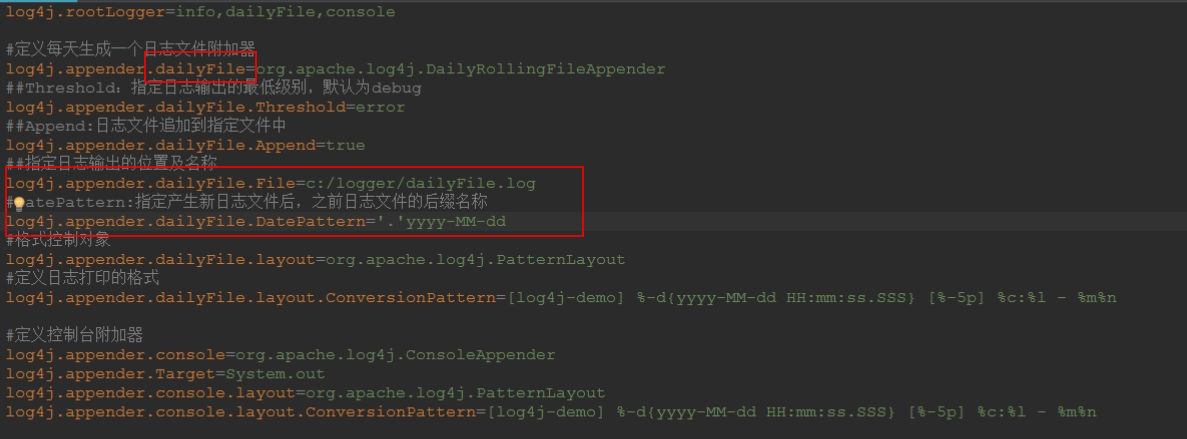


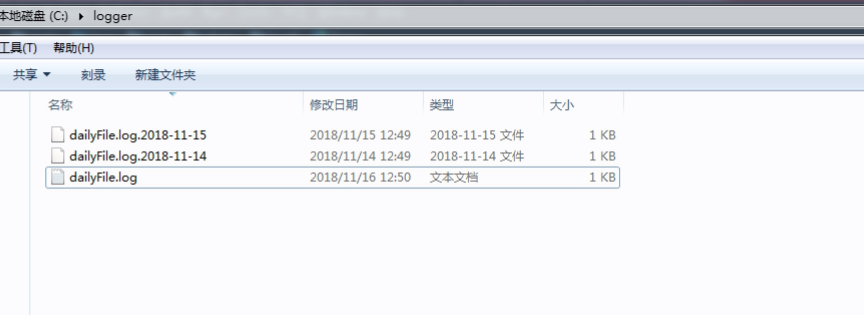
#### Append



在文件附加器中使用

#### 每天生成一个日志文件





没有带时间的是 当天的日志文件，到第二天，将当天的文件重命名为昨天的日志文件

#### 每小时生成一个日志文件

