Log4j

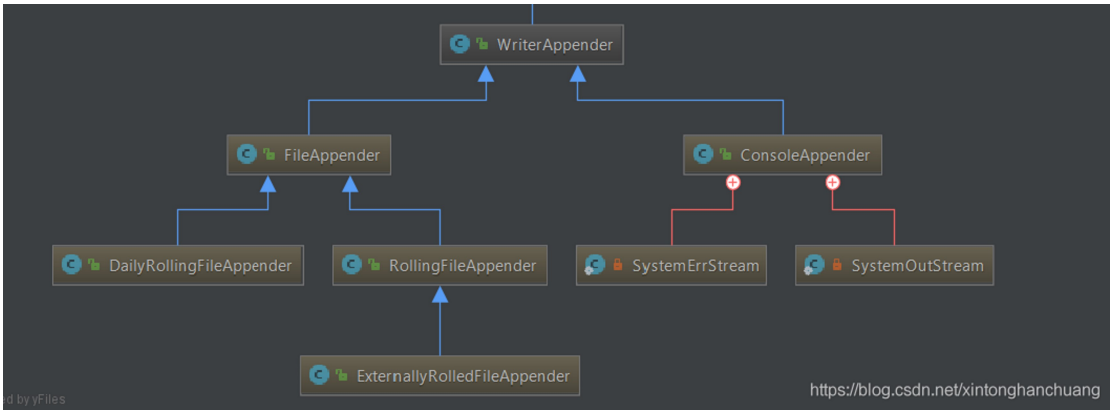
2021年6月7日 15:01

依赖

```
<dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.26</version>
    </dependency>
```

Appender

- ConsoleAppender：输出到控制台；
- FileAppender：输出到指定文件；
- DailyRollingFileAppender：每天产生一个单独的日志文件；
- RollingFileAppender：限制日志文件大小，每当达到大小限制时生成一个新的日志文件；
- WriterAppender：将日志信息以流格式发送到任意指定的地方



配置文件

Log4j 支持两种配置文件格式，一种是 XML 格式的文件，一种是 Java 特性文件（键 = 值 properties 文件）。

公共属性

属性名	属性值类型	描述
Threshold	String	日志输出级别
ImmediateFlush	Boolean	是否立即输出
Encoding	String	日志输出编码
layout	Layout	日志输出布局
ConversionPattern	String	当日志输出布局为自定义时，使用选项

不同的Appender具有其不同的属性设置：
ConsoleAppender:

属性名	属性值类型	描述
Target	System.out	输出到控制台

DailyRollingFileAppender:

属性名	属性值类型	描述
File	String	文件路径
Append	Boolean	是否尾部附加
DatePattern	String	生成周期

RollingFileAppender:

属性名	属性值类型	描述
File	String	文件路径
Append	Boolean	是否尾部附加
MaxFileSize	String	单个文件大小
MaxBackupIndex	Number	备份文件记录数

log4j.properties

```
# 格式：log4j.rootLogger = [ level ] , appenderName1, appenderName2, …
log4j.rootLogger = DEBUG, console_out, file_out, daily_file_out, html_file_out

#### ConsoleAppender:控制台输出 ####

log4j.appender.console_out = org.apache.log4j.ConsoleAppender
# 输出到控制台
log4j.appender.console_out.Target = System.out
# 指定控制台输出日志级别
log4j.appender.console_out.Threshold = WARN
# 默认值是 true, 表示是否立即输出
log4j.appender.console_out.ImmediateFlush = true
# 设置编码方式
log4j.appender.console_out.Encoding = UTF-8
# 日志输出布局
log4j.appender.console_out.layout = org.apache.log4j.PatternLayout
# 如果日志输出布局为PatternLayout 自定义级别，需要使用ConversionPattern指定输出格式
log4j.appender.console_out.layout.ConversionPattern =[%d{yyyy-MM-dd HH:mm:ss,SSS}]-[%p] -%c -%r -%
l.%M(%L) | %m%n

#### FileAppender: 输出到文件 ####

log4j.appender.file_out = org.apache.log4j.FileAppender
# 指定输出文件路径
log4j.appender.file_out.File = ./log/log4j-FA.log
# 指定输出日志级别
log4j.appender.file_out.Threshold = INFO
# 默认为true,意味着输出方式为追加，反之则为覆盖
log4j.appender.file_out.Append = true
# 默认值是 true, 表示是否立即输出
log4j.appender.file_out.ImmediateFlush = true
# 设置编码方式
```

```
log4j.appender.file_out.Encoding = UTF-8
# 日志输出布局
log4j.appender.file_out.layout = org.apache.log4j.PatternLayout
# 如果日志输出布局为PatternLayout 自定义级别，需要使用ConversionPattern指定输出格式
log4j.appender.file_out.layout.ConversionPattern =FA-[framework] [%d{yyyy-MM-dd HH:mm:ss,SSS}]-[%
p] -%c -%r -%l.%M(%L) | %m%n

#### DailyRollingFileAppender: 指定周期输出到新文件 ####

log4j.appender.daily_file_out = org.apache.log4j.DailyRollingFileAppender
# 指定输出文件路径
log4j.appender.daily_file_out.File = ./log/log4j-DFA.log
# 指定输出日志级别
log4j.appender.daily_file_out.Threshold = DEBUG
# 默认为true,意味着输出方式为追加，反之则为覆盖
log4j.appender.daily_file_out.Append = true
# 默认值是 true, 表示是否立即输出
log4j.appender.daily_file_out.ImmediateFlush = true
# 设置编码方式
log4j.appender.daily_file_out.Encoding = UTF-8
# 指定分隔周期：月，周，天，时，分
# '.': 每月
# '.'yyyy-ww: 每周
# '.'yyyy-MM-dd: 每天
# '.'yyyy-MM-dd-a: 每天两次
# '.'yyyy-MM-dd-HH: 每小时
# '.'yyyy-MM-dd-HH-mm: 每分钟
log4j.appender.daily_file_out.DatePattern ='.'yyyy-MM-dd-HH-mm
# 日志输出布局
log4j.appender.daily_file_out.layout = org.apache.log4j.PatternLayout
# 如果日志输出布局为PatternLayout 自定义级别，需要使用ConversionPattern指定输出格式
log4j.appender.daily_file_out.layout.ConversionPattern =DFA-[framework] [%d{yyyy-MM-dd
HH:mm:ss,SSS}]-[%p] -%c -%r -%l.%M(%L) | %m%n
```

ConversionPattern

当使用自定义格式输出时，需要使用ConversionPattern自定义输出格式，Log4j采用的是C语言中Printf函数的格式

%p: 输出日志信息优先级，即DEBUG，INFO，WARN，ERROR，FATAL,
%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
%r: 输出自应用启动到输出该log信息耗费的毫秒数
%c: 输出日志信息所属的类目，通常就是所在类的全名
%t: 输出产生该日志事件的线程名
%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：Log4jDemo.main(Log4jDemo.Java:23)
%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。
%%: 输出一个”%”字符
%F: 输出日志消息产生时所在的文件名称
%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

Layout

HTMLLayout：以HTML表格形式布局输出；

Log session start time Mon Jun 07 17:43:48 CST 2021

Time	Thread	Level	Category	Message
0	main	INFO	org.example.LogDemo	===info===
2	main	DEBUG	org.example.LogDemo	===debug===
2	main	WARN	org.example.LogDemo	===debug===

PatternLayout：以自定义的格式输出；

```
"C:\Program Files\Java\jdk1.8.0_211\bin\java.exe" ...
[2021-06-07 17:47:46,319]-[WARN] -org.example.LogDemo -3 -org.example.LogDemo.log(LogDemo.java:16).log(16) | ===debug===

Process finished with exit code 0
```

SimpleLayout：简单格式，是包含日志信息的级别和信息字符串；

log4j_SL.log - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
INFO - ===info===
DEBUG - ===debug===
WARN - ===debug===
```

TTCCLayout：包含日志产生的时间、线程、类别等等信息

log4jTCC.log - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
[main] INFO org.example.LogDemo - ===info===
[main] WARN org.example.LogDemo - ===debug===
[main] INFO org.example.LogDemo - ===info===
```

XML配置

把properties属性按xml节点换行拆分即可，属性使用param的value赋值

XML配置文件

登录后复制

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration  debug="true" xmlns:log4j='http://jakarta.apache.org/log4j/' >
    <!-- 输出到控制台 -->
    <appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
        <!-- 设置日志输出的样式 -->
        <layout class="org.apache.log4j.PatternLayout">
            <!-- 设置日志输出的格式 -->
            <param name="ConversionPattern" value="[%d{yyyy-MM-dd HH:mm:ss:SSS}] [%-5p] [method:%l]%n%m%n%n" />
        </layout>
        <!--过滤器设置输出的级别-->
        <filter class="org.apache.log4j.varia.LevelRangeFilter">
            <!-- 设置日志输出的最小级别 -->
            <param name="levelMin" value="WARN" />
            <!-- 设置日志输出的最大级别 -->
            <param name="levelMax" value="ERROR" />
            <!-- 设置日志输出的xxx，默认是false -->
            <param name="AcceptOnMatch" value="true" />
        </filter>
    </appender>

```

参考资料:
https://blog.csdn.net/xintonghanchuang/article/details/90905236