参考：

<http://blog.csdn.net/autfish/article/details/51203709>

<http://blog.csdn.net/u010201484/article/details/51723455> （good，别人的好文章）

<http://blog.csdn.net/seven_zhao/article/details/42172441>

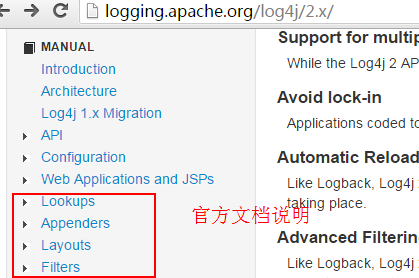
基本说明

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。

下载地址:

<http://logging.apache.org/log4j/1.2/>

<http://logging.apache.org/log4j/2.x/index.html> log4j2



添加依赖：

普通**java**项目手动添加jar包

1. log4j-api-2.5.jar
2. log4j-core-2.5.jar

Maven项目pom.xml

1. **<dependencies>**
2. **<dependency>**
3. **<groupId>**org.apache.logging.log4j**</groupId>**
4. **<artifactId>**log4j-api**</artifactId>**
5. **<version>**2.5**</version>**
6. **</dependency>**
7. **<dependency>**
8. **<groupId>**org.apache.logging.log4j**</groupId>**
9. **<artifactId>**log4j-core**</artifactId>**
10. **<version>**2.5**</version>**
11. **</dependency>**
12. **</dependencies>**

基础案例：

Java文件

**package** net.shopin.xyy;

**import** org.apache.logging.log4j.LogManager;

**import** org.apache.logging.log4j.Logger;

**public** **class** Log4jTest {

**private** **static** Logger *logger* = LogManager.*getLogger*(Log4jTest.**class**.getName());

**public** **static** **void** main(String[] args) {

*logger*.trace("trace\*\*\*\*\*\*\*");

*logger*.debug("debug\*\*\*\*\*\*\*\*\*\*\*");

*logger*.info("info\*\*\*\*\*\*\*\*\*\*\*\*");

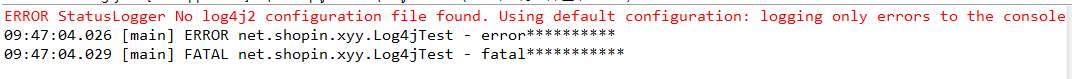
*logger*.error("error\*\*\*\*\*\*\*\*\*\*");

*logger*.fatal("fatal\*\*\*\*\*\*\*\*\*\*\*");

}

}

直接运行结果：没有配置文件



系统说，没有找到log4j2的配置文件，所有使用了默认的配置，级别为error的日志打印到控制台上。

说明：(log4j2默认会在classpath目录下寻找log4j.json、log4j.jsn、log4j2.xml等名称的文件，如果都没有找到，则会按默认配置输出，也就是输出到控制台。)

Log4j2.xml配置文件

# 测试案例一（不定义logger4j2.xml）

问题：Root的级别和appenders内的appender的级别哪一个优先级高

答案：谁定义的级别更加高，就取谁。（取其高）

Java文件

**package** net.shopin.xyy;

**import** org.apache.logging.log4j.LogManager;

**import** org.apache.logging.log4j.Logger;

**public** **class** Log4jTest {

**private** **static** Logger *logger* = LogManager.*getLogger*(Log4jTest.**class**.getName());

**public** **static** **void** main(String[] args) {

*logger*.trace("trace\*\*\*\*\*\*\*");

*logger*.debug("debug\*\*\*\*\*\*\*\*\*\*\*");

*logger*.info("info\*\*\*\*\*\*\*\*\*\*\*\*");

*logger*.error("error\*\*\*\*\*\*\*\*\*\*");

*logger*.fatal("fatal\*\*\*\*\*\*\*\*\*\*\*");

}

}

Log4j2.xml(log4j2默认会在classpath目录下寻找log4j.json、log4j.jsn、log4j2.xml等名称的文件，如果都没有找到，则会按默认配置输出，也就是输出到控制台。)

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<Configuration status=*"WARN"*>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"* />

<ThresholdFilter level=*"fatal"* onMatch=*"ACCEPT"* onMismatch=*"DENY"*/>

</Console>

</Appenders>

<Loggers>

<Root level=*"error"*>

<AppenderRef ref=*"Console"* />

</Root>

</Loggers>

</Configuration>

输出结果：



修改xml文件

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<Configuration status=*"WARN"*>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"* />

<ThresholdFilter level=*"info"* onMatch=*"ACCEPT"* onMismatch=*"DENY"*/>

</Console>

</Appenders>

<Loggers>

<Root level=*"fatal"*>

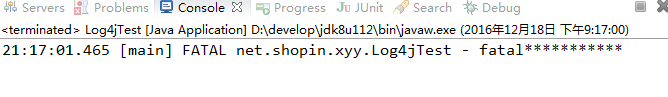
<AppenderRef ref=*"Console"* />

</Root>

</Loggers>

</Configuration>

输出结果：



结论：root的级别和appender的级别，取其高。谁定义的级别更严格，去谁的级别。一般appender不会去定义级别的。

# 案例二：（level优先级）

问题描述：

Root的级别和其他logger的级别同时定义，谁的优先级高。

修改log4j.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<Configuration status=*"WARN"*>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"* />

</Console>

</Appenders>

<Loggers>

<logger name=*"net.shopin.xyy"* level=*"debug"* additivity=*"false"*>

<appender-ref ref=*"Console"* />

</logger>

<Root level=*"fatal"*>

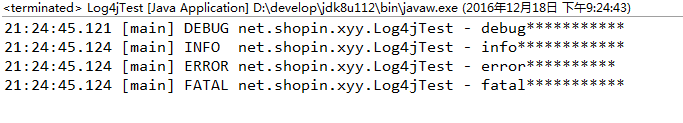
<AppenderRef ref=*"Console"* />

</Root>

</Loggers>

</Configuration>

输出结果：



修改log4j.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<Configuration status=*"WARN"*>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"* />

</Console>

</Appenders>

<Loggers>

<logger name=*"net.shopin.xyy"* level=*"fatal"* additivity=*"false"*>

<appender-ref ref=*"Console"* />

</logger>

<Root level=*"debug"*>

<AppenderRef ref=*"Console"* />

</Root>

</Loggers>

</Configuration>

输出结果：



结论：

一般定义Logger,是针对某个包及其子包。不同于root定义的级别，需要特殊处理，才会定义logger,比如编码时期，需要打印出sql语句，这是可以针对mapper.java所在的包新建一个logger,定义level为debug,(这样就可以打印sql语句)，但是其他的包不需要显示那么多详细的内容，这是root(根节点)，定义为info,那些没有特殊定义logger的包，就会默认继承root定义的level-info，只打印info级别及其以上的日志了。（log4j的继承机制）

So,logger的存在是为了单独处理某些包下日志的显示，优先级高于root,如果某个包定义了logger,就会自动忽略root定义的级别。

# 测试案例三（level级别的优先级）

问题描述：

Logger的级别和appenders下的appender的级别优先级高低

答案：案例一。取其高。一般appender还是不需要定义日志级别。在root和logger定义级别就可以了。

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<Configuration status=*"WARN"*>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"* />

<ThresholdFilter level=*"error"* onMatch=*"ACCEPT"* onMismatch=*"DENY"*/>

</Console>

</Appenders>

<Loggers>

<logger name=*"net.shopin.xyy"* level=*"fatal"* additivity=*"false"*>

<appender-ref ref=*"Console"* />

</logger>

</Loggers>

</Configuration>

结果：



修改log4j2.xml文件

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<Configuration status=*"WARN"*>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"* />

<ThresholdFilter level=*"error"* onMatch=*"ACCEPT"* onMismatch=*"DENY"*/>

</Console>

</Appenders>

<Loggers>

<logger name=*"net.shopin.xyy"* level=*"info"* additivity=*"false"*>

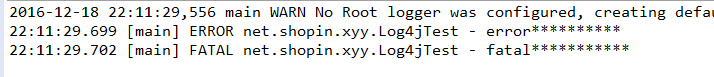
<appender-ref ref=*"Console"* />

</logger>

</Loggers>

</Configuration>

输出结果：



# 节点说明

1.问题描述：文件各个节点的说明

|  |
| --- |
| 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：  %p 输出优先级，即ALL,TRACE,DEBUG，INFO，WARN，ERROR，FATAL ,ALL （priority）  %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 (class)  %t 输出产生该日志事件的线程名 (thread)  %n 输出一个回车换行符，Windows平台为“rn”，Unix平台为“n” (换行)  %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 （date）  %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。举例：Testlog4.main(TestLog4.java:10)（location）  %m 输出代码中指定的消息(message) |

## 正确理解过滤器ThresholdFilter

<ThresholdFilter level="error" onMatch="ACCEPT" onMismatch="DENY"/>

onMatch:大于等于error的级别接受。（默认NEUTRAL）

OnMismatch:小于error的级别被拒绝。(默认DENY)

Neutral:中立状态，不接受不拒绝看其他过滤器，所有都通过，即为接受。

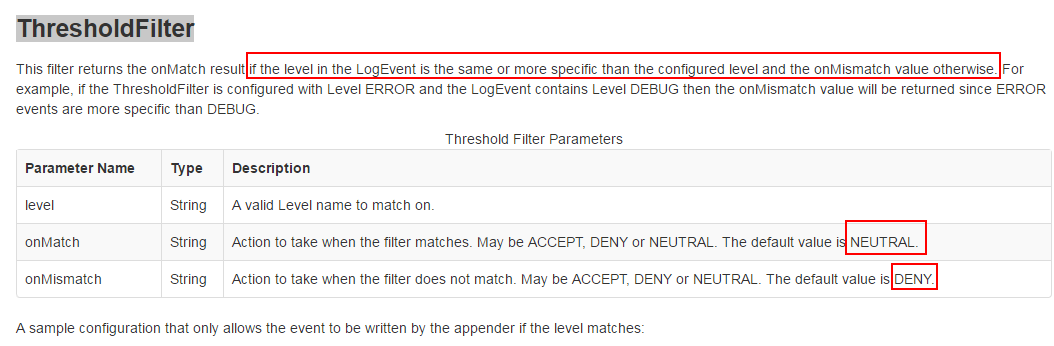
一般这么定义：>=info级别的不显示,<info中立; >=debug级别显示,<debug拒绝，综合，仅仅显示debu级别

<Filters>

<ThresholdFilter level="info" onMatch="DENY" onMismatch="NEUTRAL"/>

<ThresholdFilter level="debug" onMatch="ACCEPT" onMismatch="DENY"/>

</Filters>



  Filter

    与防火墙过滤的规则相似，log4j2的过滤器也将返回三类状态：Accept（接受）, Deny（拒绝） 或Neutral（中立）。

    其中，Accept意味着不用再调用其他过滤器了，这个LogEvent将被执行；

    Deny意味着马上忽略这个event，并将此event的控制权交还给过滤器的调用者；

Neutral则意味着这个event应该传递给别的过滤器，如果再没有别的过滤器可以传递了，那么就由现在这个过滤器来处理。

## 正确理解 %-4relative 左对齐，最小宽度为4

官网说明：<http://logging.apache.org/log4j/2.x/manual/layouts.html#PatternLayout>

<pattern>里面的转换符说明：

<PatternLayout pattern=*"%d{DEFAULT} [%t] %-4level %logger - %msg%n"* />

%level == %p %logger == %c

|  |  |
| --- | --- |
| **转换符** | **作用** |
| **c** {*length* }  **lo** {*length* }  **logger** {*length* } | 输出日志的logger名，可有一个整形参数，功能是缩短logger名，设置为0表示只输入logger最右边点符号之后的字符串。   |  |  |  | | --- | --- | --- | | **Conversion specifier** | **Logger name** | **Result** | | %logger | mainPackage.sub.sample.Bar | mainPackage.sub.sample.Bar | | %logger{0} | mainPackage.sub.sample.Bar | Bar | | %logger{5} | mainPackage.sub.sample.Bar | m.s.s.Bar | | %logger{10} | mainPackage.sub.sample.Bar | m.s.s.Bar | | %logger{15} | mainPackage.sub.sample.Bar | m.s.sample.Bar | | %logger{16} | mainPackage.sub.sample.Bar | m.sub.sample.Bar | | %logger{26} | mainPackage.sub.sample.Bar | mainPackage.sub.sample.Bar | |
| **C** {*length* }  **class** {*length* } | 输出执行记录请求的调用者的全限定名。参数与上面的一样。尽量避免使用，除非执行速度不造成任何问题。 |
| **contextName**  **cn** | 输出上下文名称。 |
| **d** {*pattern* }  **date** {*pattern* } | 输出日志的打印日志，模式语法与java.text.SimpleDateFormat 兼容。   |  |  | | --- | --- | | **Conversion Pattern** | **Result** | | %d | 2006-10-20 14:06:49,812 | | %date | 2006-10-20 14:06:49,812 | | %date{ISO8601} | 2006-10-20 14:06:49,812 | | %date{HH:mm:ss.SSS} | 14:06:49.812 | | %date{dd MMM yyyy ;HH:mm:ss.SSS} | 20 oct. 2006;14:06:49.812 | |
| **F / file** | 输出执行记录请求的java源文件名。尽量避免使用，除非执行速度不造成任何问题。 |
| **caller{depth}caller{depth, evaluator-1, ... evaluator-n}** | 输出生成日志的调用者的位置信息，整数选项表示输出信息深度。  例如， **%caller{2}**   输出为：  0 [main] DEBUG - logging statement  Caller+0 at mainPackage.sub.sample.Bar.sampleMethodName(Bar.java:22)  Caller+1 at mainPackage.sub.sample.Bar.createLoggingRequest(Bar.java:17)  例如， **%caller{3}**   输出为：  16 [main] DEBUG - logging statement  Caller+0 at mainPackage.sub.sample.Bar.sampleMethodName(Bar.java:22)  Caller+1 at mainPackage.sub.sample.Bar.createLoggingRequest(Bar.java:17)  Caller+2 at mainPackage.ConfigTester.main(ConfigTester.java:38) |
| **L / line** | 输出执行日志请求的行号。尽量避免使用，除非执行速度不造成任何问题。 |
| **m / msg / message** | 输出应用程序提供的信息。 |
| **M / method** | 输出执行日志请求的方法名。尽量避免使用，除非执行速度不造成任何问题。 |
| **n** | 输出平台先关的分行符“\n”或者“\r\n”。 |
| **p / le / level** | 输出日志级别。 |
| **r / relative** | 输出从程序启动到创建日志记录的时间，单位是毫秒 |
| **t / thread** | 输出产生日志的线程名。 |
| **replace(*p* ){r, t}** | **p** 为日志内容，**r** 是正则表达式，将**p** 中符合**r** 的内容替换为**t** 。  例如， "%replace(%msg){'\s', ''}" |

**格式修饰符(**optional format modifiers**)，与转换符(**conversion specifier**)共同使用：**

可选的格式修饰符位于“%”和转换符之间。

@可选修饰符**左对齐** 标志，符号是减号“-”；

@可选的**最小宽度** 修饰符，用十进制数表示。

如果字符小于最小宽度，则左填充或右填充，默认是左填充（即右对齐），填充符为空格。

如果字符大于最小宽度，字符永远不会被截断。

**@最大宽度** 修饰符，符号是点号"."后面加十进制数。

如果字符大于最大宽度，则从前面截断。

点符号“.”后面加减号“-”在加数字，表示从尾部截断。

例如：%-4relative 表示，将输出从程序启动到创建日志记录的时间 进行左对齐 且最小宽度为4。

# 测试案例五（additivity=”true”）

问题描述：针对logger 的属性additivity=*"false"和*additivity=*"true"的区别*以及为true时候，父层的级别是否起到作用。

Log4j2.xml的配置

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!-- status :表示log4j自身日志的打印级别 -->

<Configuration status=*"warn"*>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<!-- <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" /> -->

<PatternLayout pattern=*"[%d{HH:mm:ss:SSS}] [%p] - %l - %m%n"*/>

<!-- <ThresholdFilter level="error" onMatch="ACCEPT" onMismatch="DENY"/> -->

</Console>

<!-- append=false,表示不追加，下次重启的时候回清空此文档，可用于测试。 -->

<File name=*"test"* fileName=*"D:/logs/app.log"* append=*"false"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"* />

</File>

</Appenders>

<Loggers>

<!-- additivity="false"表示在该logger中输出的日志不会再延伸到父层logger。这里如果改为true，则会延伸到Root Logger，遵循Root Logger的配置也输出一次。 -->

<logger name=*"net.shopin.xyy"* level=*"trace"* additivity=*"true"*>

<appender-ref ref=*"test"* />

</logger>

<Root level=*"error"*>

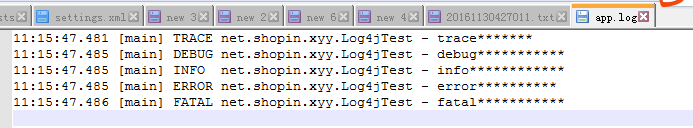
<AppenderRef ref=*"Console"* />

</Root>

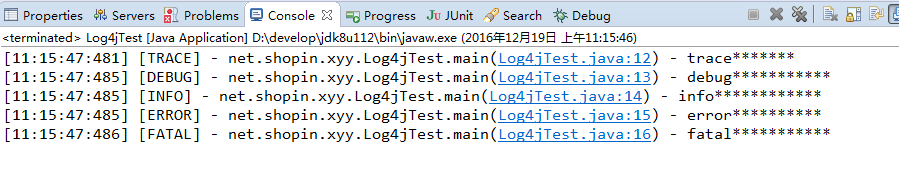
</Loggers>

</Configuration>

打印文件app.log内容：



Console打印信息：



发现，root的level是error,但是控制台依旧打印了trace级别的信息。即打印的内容没有看fu层的级别，而是遵守子层定义的级别规范。

原理（copy自网络）：<http://blog.csdn.net/jiafu1115/article/details/17582411>

|  |
| --- |
| log打印的规则：         当某个logger的logging request(即printing method(error(),info()..))的级别高于或者等于该logger的级别(即在log4j.properties或者log4j.xml中定义)的时候，该logging request就为enable.          一旦该logger的logging request为enable，那么该logging request将总会打印到该logger所有的appender中包括它的所有父logger的appender。（而不会管父logger的级别如何）。除非该logger的父logger的additivity设置为false，默认的情况下为true。         某个logger的additivity设置为false，表示log只打印到本log的appender中，而不再打印到其父logger的appender。 |

解释说明：

Additivity=true,表示会打印到父logger的appender中。

Additivity=false, 表示不会打印到父logger的appender中。

1. 只要编码中的级别大于等于log4j2.xml配置的级别（eg.上面配置文件的logger: *net.shopin.xyy,配置的级别为trace,那么在net.shopin.xyy包和子包编码是凡是调用方法trace(),debug();info()….*）的logger都是enable。
2. 凡是enable的logger,就会打印在自己logger的所有appenders以及所有父logger的所有appenders（而不管父logger定义的级别）。除非父logger的additivity设置为false（表示不打印到父logger的appender中，即向上传递停止）,默认值为true.

So，这里根节点root定义级别为error,依然显示trace的内容（因为logger: *net.shopin.xyy的级别为trace*）

修改log4j2.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!-- status :表示log4j自身日志的打印级别 -->

<Configuration status=*"warn"*>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<!-- <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" /> -->

<PatternLayout pattern=*"[%d{HH:mm:ss:SSS}] [%p] - %l - %m%n"*/>

<!-- <ThresholdFilter level="error" onMatch="ACCEPT" onMismatch="DENY"/> -->

</Console>

<!-- append=false,表示不追加，下次重启的时候回清空此文档，可用于测试。 -->

<File name=*"test"* fileName=*"D:/logs/app.log"* append=*"false"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"* />

</File>

<File name=*"testChild"* fileName=*"D:/logs/appChild.log"* append=*"false"*>

<PatternLayout pattern=*"%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"* />

</File>

</Appenders>

<Loggers>

<!-- additivity="false"表示在该logger中输出的日志不会再延伸到父层logger。这里如果改为true，则会延伸到Root Logger，遵循Root Logger的配置也输出一次。 -->

<logger name=*"net.shopin.xyy"* level=*"info"* additivity=*"false"*>

<AppenderRef ref=*"test"* />

</logger>

<logger name=*"net.shopin.xyy.temp"* level=*"error"* additivity=*"true"*>

<appender-ref ref=*"testChild"* />

</logger>

<Root level=*"error"*>

<AppenderRef ref=*"Console"* />

</Root>

</Loggers>

</Configuration>

运行结果：

*appChild.log：*

*12:01:11.761 [main] ERROR net.shopin.xyy.temp.Log4jTesttChild - error\*\*\*\*\*\*\*\*\*\**

*12:01:11.764 [main] FATAL net.shopin.xyy.temp.Log4jTesttChild - fatal\*\*\*\*\*\*\*\*\*\*\**

*app.log：*

*12:01:11.761 [main] ERROR net.shopin.xyy.temp.Log4jTesttChild - error\*\*\*\*\*\*\*\*\*\**

*12:01:11.764 [main] FATAL net.shopin.xyy.temp.Log4jTesttChild - fatal\*\*\*\*\*\*\*\*\*\*\**

*控制台：*

无打印信息。

说明：

*net.shopin.xyy.temp子包定义error，传递到xyy父包，级别显示为error,因为xyy包的*additivity=*"false，所以不传递到root,控制台不显示信息。*

# 项目可直接使用的例子

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!-- status :表示log4j自身日志的打印级别 -->

<!-- monitorInterval:含义是每隔300秒重新读取配置文件，可以不重启应用的情况下修改配置 -->

<Configuration status=*"warn"* monitorInterval=*"300"*>

<properties>

<property name=*"INFO\_FILE\_NAME"*>info</property>

<property name=*"ERROR\_FILE\_NAME"*>error</property>

</properties>

<Appenders>

<Console name=*"Console"* target=*"SYSTEM\_OUT"*>

<!-- 左对齐，最小宽度为4；长度>10也可正常显示，不足4用空格补齐 -->

<PatternLayout pattern=*"%d{DEFAULT} [%t] %-4level - %l - %msg%n"* />

</Console>

<!-- fileName:日志位置以及文件名；filePattern:rolling时新建文件的位置以及命名规则。命名文件名称需要细到时分秒是注意不要这么写

HH:mm:ss，文件名称不可以包含特殊字符：使用"-"代替 -->

<RollingRandomAccessFile name=*"INFO\_FILE"*

fileName=*"${sys:user.home}/logs/jx\_padmg/${INFO\_FILE\_NAME}.log"*

filePattern=*"${sys:user.home}/logs/jx\_padmg/base/$${date:yyyy-MM}/${INFO\_FILE\_NAME}-%d{yyyy-MM-dd}-%i.log.gz"* append=*"true"*>

<Filters>

<ThresholdFilter level=*"WARN"* onMatch=*"DENY"* onMismatch=*"NEUTRAL"* />

<ThresholdFilter level=*"INFO"* onMatch=*"ACCEPT"* onMismatch=*"DENY"* />

</Filters>

<PatternLayout

pattern=*"%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %l - %msg%n"* />

<Policies>

<!-- 下面两个策略：满足一个，即会产生一个文件 -->

<!-- 日志文件大于100M,就新建文件 -->

<SizeBasedTriggeringPolicy size=*"100 MB"* />

<!-- 结合filePattern:精确到dd(天)，所以表示每天产生一个日志文件 -->

<TimeBasedTriggeringPolicy interval=*"1"* />

</Policies>

<!-- 作用于filePattern中的i，最大20个文件。 -->

<DefaultRolloverStrategy max=*"20"* />

</RollingRandomAccessFile>

<RollingRandomAccessFile name=*"ERROR\_FILE"*

fileName=*"${sys:user.home}/logs/jx\_padmg/${ERROR\_FILE\_NAME}.log"*

filePattern=*"${sys:user.home}/logs/jx\_padmg/error/$${date:yyyy-MM}/${BASE\_FILE\_NAME}-%d{yyyy-MM}-%i.log.gz"* append=*"true"*>

<Filters>

<ThresholdFilter level=*"fatal"* onMatch=*"DENY"* onMismatch=*"NEUTRAL"* />

<ThresholdFilter level=*"error"* onMatch=*"ACCEPT"* onMismatch=*"DENY"* />

</Filters>

<PatternLayout

pattern=*"%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %l - %msg%n"* />

<Policies>

<SizeBasedTriggeringPolicy size=*"100 MB"* />

<TimeBasedTriggeringPolicy interval=*"1"* />

</Policies>

<DefaultRolloverStrategy max=*"20"* />

</RollingRandomAccessFile>

</Appenders>

<Loggers>

<Root level=*"INFO"*>

<AppenderRef ref=*"Console"* />

<AppenderRef ref=*"INFO\_FILE"* />

<AppenderRef ref=*"ERROR\_FILE"* />

</Root>

</Loggers>

</Configuration>