# Log4j 2简介

Log4j的1.x版本已经被广泛使用于很多应用程序中。然而，它这些年的发展已经放缓。它变得越来越难以维护，因为它需要严格遵循很老的Java版本，并在2015年8月寿终正寝。

它的替代品，SLF4J和Logback对框架做了很多必要的改进。

那么为什么还要费心去做Log4j 2呢？几个原因如下：

Log4j 2被设计为可以作为审计框架使用。Log4j 1.x和Logback都会在重新配置的时候失去事件，而Log4j2不会。在Logback中，Appender当中的异常对应用从来都是不可见的。

但Log4j2的Appender可以设置为允许将异常渗透给应用程序。

Log4j 2包含基于LMAX Disruptor库的下一代异步日志器。在多线程情况下，异步日志器具有比Log4j 1.x和Logback高出10倍的吞吐性能以及更低的延迟。

Log4j 2在稳定记录状态下，对单机应用是无垃圾的，对Web应用是低垃圾的。这不仅降低了垃圾回收器的压力，还可以提供更好的响应性能。

Log4j 2使用插件系统使得它非常容易通过新的Appender、Filter、Layout、Lookup和Pattern Converter来扩展框架，且不需要对Log4j做任何修改。

由于插件系统的配置更简单了，配置项不需要声明类名称。

支持自定义日志级别。自定义日志级别可以在代码或配置中定义。

支持Lambda表达式。运行在Java 8上的客户端代码可以使用Lambda表达式来实现仅在对应的日志级别启用时延迟构造日志消息。由于不需要明确地层层把关，这带来了更简洁的代码。

支持Message对象。Message允许支持感兴趣或复杂的结构体在日志系统中传输，且可以被高效地操作。用户可以自由地创建他们自己的Message类型，并编写自定义的Layout、Filter

和Lookup来操作它们。

Log4j 1.x支持Appender上的Filter。Logback引入了TurboFilter来在事件被Logger处理之前对它们进行过滤。Log4j 2支持的Filter可以设置为在被Logger

接管之前即处理事件，如同它在Logger或Appender中被处理。

很多Logback的Appender不接受一个Layout，且只能发送固定格式的数据。而大多数Log4j 2的Appender接受Layout，允许数据以任意一种所需的格式传输。

Log4j 1.x和Logback中的Layout返回一个String。这导致了在Logback Encoder中讨论的问题。Log4j 2用更简单的方法，Layout总是返回一个字节数组。

优点是这意味着它们可以用于任何Appender，而不仅仅是写入到OutputStream中的那些。

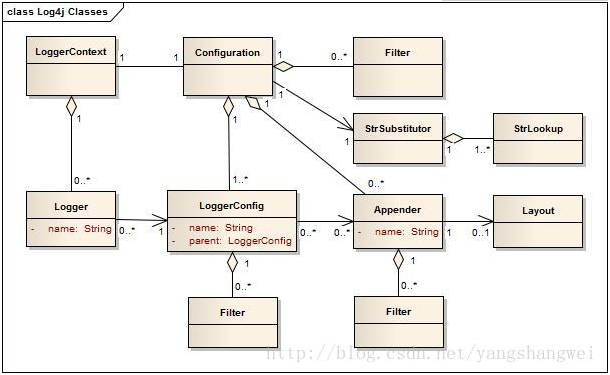
Syslog Appender既支持TCP也支持UDP，同样支持BSD系统日志以及RFC 5424格式。

Log4j 2利用了Java 5的并发优势，并在尽可能最低的程度上进行锁定。Log4j 1.x中已知存在死锁问题。其中很多已经在Logback中修复，但很多Logback的class文件仍然需要

在更高的编译级别中同步。

这是一个被所有ASF项目集体支持使用的Apache软件基金会项目。如果你想要贡献或修改，只要参照贡献中的方法。

**架构**



应用程序要使用Log4j 2的API，需要从LogManager中获取一个有明确名称的Logger。

LogManager将会定位到一个合适的LoggerContext并且从中获取Logger。

如果Logger必须被创建，那么它会和包含这些信息的LogConfig相关联：

a）与Logger相同的名称；

b）父包的名称；

c）根LoggerConfig。LoggerConfig对象根据配置中的Logger声明而创建。

LoggerConfig与实际处理LogEvent事件的Appender关联。

# 日志级别



在表格中，垂直列为LogEvent的级别，水平列为从合适的LoggerConfig中分配到的级别。二者的交点处标识了LogEvent是否会被通过并传递给下一步处理，是(YES)或否(NO)。

# Filter

Log4j提供Filter并可应用于：控制被传递到任何LoggerConfig之前、控制被传递到达一个LoggerConfig但在调用任何Appender之前、控制被传递到一个LoggerConfig单在调用一个指定的Appender和每一个Appender之前。

与防火墙过滤的方式类似，每一个Filter都将返回三个结果之一：Accept（接受）、Deny（拒绝）或Neutral（中立）。

* 响应Accept意味着其他的Filter都不应该再被调用，而事件应该被处理。
* 响应Deny意味着事件应该被立即忽略，且将控制讲给调用处。
* 响应Neutral代表事件应该被传递给其他的Filter。如果没有其他Filter，则事件将被处理。

尽管一个事件可能被Filter接受，但事件仍然可能不被记录。这种情况会发生于事件被LoggerConfig之前的Filter接受，但被LoggerConfig的Filter拒绝或者被所有的Appender拒绝。

# Appender 将日志请求打印到多个目标

<http://logging.apache.org/log4j/2.x/manual/appenders.html>

Log4j允许将日志请求打印到多个目标。用Log4j的说法，一个输出的目标位置被称为Appender。目前，Appender存在几种：控制台、文件、远程Socket服务器、Apache Flume、JMS、远程UNIX系统日志后台以及好几种数据库API。

一个Logger上可以装配多个Appender。

# Layout 自定义输出格式

用户不仅希望自定义输出的目的位置，也希望自定义输出格式。

这可以通过将一个Layout与Appender关联来实现。Layout负责根据用户的希望来格式化LogEvent，然而是Appender负责将格式化的内容输出到目的位置。

PatternLayout，Log4j中的一部分，让用户根据C语言printf函数的方式来具体化输出格式。

例如，使用转换模式“%r [%t] %-5p %c - %m%n”的PatternLayout将会输出类似于下面的内容：

176 [main] INFO org.foo.Bar - Located nearest gas station.

第一个字段是程序启动以来锁经过的毫秒时间。   
第二个字段是发出日志请求的线程。   
第三个字段是日志声明的级别。   
第四个字段是与日志请求相关联的Logger名称。

在“-”之后的文本是日志的消息内容。

Log4j带有很多不同的Layout以支持诸如JSON、XML、HTML和Syslog

# 转到Log4j 2 API

大多数情况下，从Log4j 1.x API转换到Log4j 2相当简单。很多日志声明都不需要修改，但以下这些变更是必要的



## 控制台Appender的简单配置

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

<Configuration>

<Appenders>

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

<PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>

</Console>

</Appenders>

<Loggers>

<Logger name="org.apache.log4j.xml" level="info"/>

<Root level="debug">

<AppenderRef ref="STDOUT"/>

</Root>

</Loggers>

</Configuration>

## 文件Appender的简单配置

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

<Configuration>

<Appenders>

<File name="A1" fileName="A1.log" append="false">

<PatternLayout pattern="%t %-5p %c{2} - %m%n"/>

</File>

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

<PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>

</Console>

</Appenders>

<Loggers>

<Logger name="org.apache.log4j.xml" level="debug">

<AppenderRef ref="A1"/>

</Logger>

<Root level="debug">

<AppenderRef ref="STDOUT"/>

</Root>

</Loggers>

</Configuration>

## SocketAppender

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

<Configuration>

<Appenders>

<Socket name="A1" host="localHost" port="5000">

<SerializedLayout/>

</Socket>

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

<PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>

</Console>

</Appenders>

<Loggers>

<Logger name="org.apache.log4j.xml" level="debug">

<AppenderRef ref="A1"/>

</Logger>

<Root level="debug">

<AppenderRef ref="STDOUT"/>

</Root>

</Loggers>

</Configuration>

## AsyncAppender

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

<Configuration status="debug">

<Appenders>

<File name="TEMP" fileName="temp">

<PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>

</File>

<Async name="ASYNC">

<AppenderRef ref="TEMP"/>

</Async>

</Appenders>

<Loggers>

<Root level="debug">

<AppenderRef ref="ASYNC"/>

</Root>

</Loggers>

</Configuration>

## 控制台和文件的AsyncAppender

注意AsyncAppender应该在它引用Appender的后面被配置，这会让它正确地关闭。

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

<Configuration status="debug">

<Appenders>

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

<PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>

</Console>

<File name="TEMP" fileName="temp">

<PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>

</File>

<Async name="ASYNC">

<AppenderRef ref="TEMP"/>

<AppenderRef ref="CONSOLE"/>

</Async>

</Appenders>

<Loggers>

<Root level="debug">

<AppenderRef ref="ASYNC"/>

</Root>

</Loggers>

</Configuration>

# 配置Log4j 2

Log4j 2的配置可以通过以下4种方式之一完成：

1. 通过以XML、JSON、YAML或属性格式编写的配置文件。
2. 以编程方式，通过创建ConfigurationFactory和配置实现。
3. 通过调用配置接口中公开的API，以编程方式将组件添加到默认配置。
4. 以编程方式，通过调用内部Logger类上的方法

举例，具体请根据实际需要修改

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

<!-- log4j2使用说明：

使用方式如下：

private static final Logger logger = LogManager.getLogger(实际类名.class.getName());

2、日志说明：

（1）请根据实际情况配置各项参数

（2）需要注意日志文件备份数和日志文件大小，注意预留目录空间

（3）实际部署的时候backupFilePatch变量需要修改成linux目录

-->

<configuration status="debug">

<Properties>

<Property name="fileName">loginModule.log</Property>

<Property name="backupFilePatch">D:/workspace/workspace-jee/HelloSpring/hello-spring4/log/</Property>

</Properties>

<!--先定义所有的appender-->

<appenders>

<!--这个输出控制台的配置-->

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

<!--控制台只输出level及以上级别的信息（onMatch），其他的直接拒绝（onMismatch）-->

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

<!-- 输出日志的格式-->

<PatternLayout pattern="%d{HH:mm:ss.SSS} %-5level %class{36} %L %M - %msg%xEx%n" />

</Console>

<!--这个会打印出所有的信息，每次大小超过size，则这size大小的日志会自动存入按年份-月份建立的文件夹下面并进行压缩，作为存档-->

<RollingFile name="RollingFile" fileName="${backupFilePatch}${fileName}"

filePattern="${backupFilePatch}$${date:yyyy-MM}/app-%d{yyyyMMddHHmmssSSS}.log.gz">

<PatternLayout

pattern="%d{yyyy.MM.dd 'at' HH:mm:ss.SSS z} %-5level %class{36} %L %M - %msg%xEx%n" />

<!-- 日志文件大小 -->

<SizeBasedTriggeringPolicy size="20MB" />

<!-- 最多保留文件数 -->

<DefaultRolloverStrategy max="20"/>

</RollingFile>

</appenders>

<!--然后定义logger，只有定义了logger并引入的appender，appender才会生效-->

<loggers>

<!--建立一个默认的root的logger-->

<Logger name="org.apache.log4j.xml" level="trace"

additivity="true">

<AppenderRef ref="RollingFile" />

</Logger>

<Root level="error">

<AppenderRef ref="Console" />

</Root>

</loggers>

</configuration>