自：https://www.jianshu.com/p/9f0c67facbe2

**Log4j2简介和异步日志梳理**

[](https://www.jianshu.com/u/c18b8379e479)

[sheen口开河](https://www.jianshu.com/u/c18b8379e479) 关注

2018.05.31 14:43\* 字数 1912 阅读 391评论 0喜欢 2

## 简介

Apache Log4j 2 is an upgrade to Log4j that provides significant improvements over its predecessor, Log4j 1.x, and provides many of the improvements available in Logback while fixing some inherent problems in Logback’s architecture.

log4j2是log4j 1.x 的升级版，参考了logback的一些优秀的设计，并且修复了一些问题，因此带来了一些重大的提升，主要有：

主要特点

* 异常处理，在logback中，Appender中的异常不会被应用感知到，但是在log4j2中，提供了一些异常处理机制。
* 性能提升， log4j2相较于log4j 1和logback都具有很明显的性能提升，后面会有官方测试的数据。
* 自动重载配置，参考了logback的设计，当然会提供自动刷新参数配置，最实用的就是我们在生产上可以动态的修改日志的级别而不需要重启应用——那对监控来说，是非常敏感的。
* 无垃圾机制，log4j2在大部分情况下，都可以使用其设计的一套无垃圾机制，避免频繁的日志收集导致的jvm gc。

### 一些概念

之前看官方文档摘抄了一些概念，这里懒得翻译了，使用log4j的都应该清楚，这里只是mark下。

log4j2 中的一些概念

#### 举个栗子

<Configuration status="debug" packages="org.apache.logging.log4j.test">

<Properties>

<Property name="filename">target/test.log</Property>

</Properties>

<Filter type="ThresholdFilter" level="trace"/>

<Appenders>

<Appender type="Console" name="STDOUT">

<Layout type="PatternLayout" pattern="%m MDC%X%n"/>

<Filters>

<Filter type="MarkerFilter" marker="FLOW" onMatch="DENY" onMismatch="NEUTRAL"/>

<Filter type="MarkerFilter" marker="EXCEPTION" onMatch="DENY" onMismatch="ACCEPT"/>

</Filters>

</Appender>

<Appender type="Console" name="FLOW">

<Layout type="PatternLayout" pattern="%C{1}.%M %m %ex%n"/>*<!-- class and line number -->*

<Filters>

<Filter type="MarkerFilter" marker="FLOW" onMatch="ACCEPT" onMismatch="NEUTRAL"/>

<Filter type="MarkerFilter" marker="EXCEPTION" onMatch="ACCEPT" onMismatch="DENY"/>

</Filters>

</Appender>

<Appender type="File" name="File" fileName="${filename}">

<Layout type="PatternLayout">

<Pattern>%d %p %C{1.} [%t] %m%n</Pattern>

</Layout>

</Appender>

</Appenders>

<Loggers>

<Logger name="org.apache.logging.log4j.test1" level="debug" additivity="false">

<Filter type="ThreadContextMapFilter">

<KeyValuePair key="test" value="123"/>

</Filter>

<AppenderRef ref="STDOUT"/>

</Logger>

<Logger name="org.apache.logging.log4j.test2" level="debug" additivity="false">

<AppenderRef ref="File"/>

</Logger>

<Root level="trace">

<AppenderRef ref="STDOUT"/>

</Root>

</Loggers>

</Configuration>

## 异步日志

log4j2最大的特点就是异步日志，其性能的提升主要也是从异步日志中受益，我们来看看如何使用log4j2的异步日志。

Log4j2提供了两种实现日志的方式，一个是通过AsyncAppender，一个是通过AsyncLogger，分别对应前面我们说的Appender组件和Logger组件。注意这是两种不同的实现方式，在设计和源码上都是不同的体现。

### AsyncAppender方式

The AsyncAppender accepts references to other Appenders and causes LogEvents to be written to them on a separate Thread. Note that exceptions while writing to those Appenders will be hidden from the application. The AsyncAppender should be configured after the appenders it references to allow it to shut down properly.  
By default, AsyncAppender uses java.util.concurrent.ArrayBlockingQueue which does not require any external libraries. Note that multi-threaded applications should exercise care when using this appender as such: the blocking queue is susceptible to lock contention and our tests showed performance may become worse when more threads are logging concurrently. Consider using lock-free Async Loggers for optimal performance.

AsyncAppender是通过引用别的Appender来实现的，当有日志事件到达时，会开启另外一个线程来处理它们。需要注意的是，如果在Appender的时候出现异常，对应用来说是无法感知的。 AsyncAppender应该在它引用的Appender之后配置，默认使用 java.util.concurrent.ArrayBlockingQueue实现而不需要其它外部的类库。 当使用此Appender的时候，在多线程的环境下需要注意，阻塞队列容易受到锁争用的影响，这可能会对性能产生影响。这时候，我们应该考虑使用无所的异步记录器（AsyncLogger）。

#### 举个栗子

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

<Configuration status="warn" name="MyApp" packages="">

<Appenders>

<File name="MyFile" fileName="logs/app.log">

<PatternLayout>

<Pattern>%d %p %c{1.} [%t] %m%n</Pattern>

</PatternLayout>

</File>

<Async name="Async">

<AppenderRef ref="MyFile"/>

</Async>

</Appenders>

<Loggers>

<Root level="error">

<AppenderRef ref="Async"/>

</Root>

</Loggers>

</Configuration>

AsyncAppender有一些配置项，如下：

image.png

除此之外还有一些其他的细节，如果感兴趣可以参考官网文档，这里就不一一列举了。

### AsyncLogger方式

AsyncLogger才是log4j2 的重头戏，也是官方推荐的异步方式。它可以使得调用Logger.log返回的更快。你可以有两种选择：全局异步和混合异步。

* **全局异步**就是，所有的日志都异步的记录，在配置文件上不用做任何改动，只需要在jvm启动的时候增加一个参数；
* **混合异步**就是，你可以在应用中同时使用同步日志和异步日志，这使得日志的配置方式更加灵活。因为Log4j文档中也说了，虽然Log4j2提供以一套异常处理机制，可以覆盖大部分的状态，但是还是会有一小部分的特殊情况是无法完全处理的，比如我们如果是记录审计日志，那么官方就推荐使用同步日志的方式，而对于其他的一些仅仅是记录一个程序日志的地方，使用异步日志将大幅提升性能，减少对应用本身的影响。混合异步的方式需要通过修改配置文件来实现，使用AsyncLogger标记配置。

#### 举个栗子

**全局异步**

配置文件不用动：

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

*<!-- Don't forget to set system property*

*-Dlog4j2.contextSelector=org.apache.logging.log4j.core.async.AsyncLoggerContextSelector*

*to make all loggers asynchronous. -->*

<Configuration status="WARN">

<Appenders>

*<!-- Async Loggers will auto-flush in batches, so switch off immediateFlush. -->*

<RandomAccessFile name="RandomAccessFile" fileName="async.log" immediateFlush="false" append="false">

<PatternLayout>

<Pattern>%d %p %c{1.} [%t] %m %ex%n</Pattern>

</PatternLayout>

</RandomAccessFile>

</Appenders>

<Loggers>

<Root level="info" includeLocation="false">

<AppenderRef ref="RandomAccessFile"/>

</Root>

</Loggers>

</Configuration>

在系统初始化的时候，增加全局参数配置：

System.setProperty("log4j2.contextSelector, "org.apache.logging.log4j.core.async.AsyncLoggerContextSelector");

你可以在你第一次获取Logger之前设置，也可以加载JVM启动参数里，类似

java -Dog4j2.contextSelector=org.apache.logging.log4j.core.async.AsyncLoggerContextSelector

**混合异步**

混合异步只需要修改配置文件即可：

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

*<!-- No need to set system property "log4j2.contextSelector" to any value*

*when using <asyncLogger> or <asyncRoot>. -->*

<Configuration status="WARN">

<Appenders>

*<!-- Async Loggers will auto-flush in batches, so switch off immediateFlush. -->*

<RandomAccessFile name="RandomAccessFile" fileName="asyncWithLocation.log"

immediateFlush="false" append="false">

<PatternLayout>

<Pattern>%d %p %class{1.} [%t] %location %m %ex%n</Pattern>

</PatternLayout>

</RandomAccessFile>

</Appenders>

<Loggers>

*<!-- pattern layout actually uses location, so we need to include it -->*

<AsyncLogger name="com.foo.Bar" level="trace" includeLocation="true">

<AppenderRef ref="RandomAccessFile"/>

</AsyncLogger>

<Root level="info" includeLocation="true">

<AppenderRef ref="RandomAccessFile"/>

</Root>

</Loggers>

</Configuration>

在上面示例的配置中，root logger就是同步的，但是com.foo.Bar的logger就是异步的。

### 使用Log4j日志的注意事项

在使用异步日志的时候需要注意一些事项，如下：

1. 不要同时使用AsyncAppender和AsyncLogger，也就是在配置中不要在配置Appender的时候，使用Async标识的同时，又配置AsyncLogger，这不会报错，但是对于性能提升没有任何好处。
2. 不要在开启了全局同步的情况下，仍然使用AsyncAppender和AsyncLogger。这和上一条是同一个意思，也就是说，如果使用异步日志，AsyncAppender、AsyncLogger和全局日志，不要同时出现。
3. 如果不是十分必须，不管是同步异步，都设置immediateFlush为false，这会对性能提升有很大帮助。

4、如果不是确实需要，不要打印location信息，比如HTML的location，或者pattern模式里的%C or $class, %F or %file, %l or %location, %L or %line, %M or %method, 等，因为Log4j需要在打印日志的时候做一次栈的快照才能获取这些信息，这对于性能来说是个极大的损耗。

## 性能提升

关于性能测试，大家可以直奔官网，哪里有很详细的数据，这里给个图：

image.png

image.png

虽然我测下来，在immediateFlush设置为false的情况下，同步异步差不了多少，但可能是我的测试条件不符合官方的，从设计和原理上来说，异步日志，无疑是个最优的选择。

## 小结

总的来说，看了一遍log4j的官网文档，对日志系统有了个比较全面的了解，以前只是copy配置来改改，没关注过很多细节，这次算是扫盲了一次。文章也只是做了个介绍，在实际使用中，还是要细细研究下配置。

另外，个人觉得异步模式无非就是在原来同步写盘的前提下，增加消息队列作为缓存，或者交个另一个线程去做，这理论上除了带来一些额外的，较小的cpu和内存的开销，应该会在高流量的时候带来不小的性能提升，对比下来，log4j2无疑是当下最值得使用的日志组件来，且可以使用其异步模式。

当然了，也不能说异步就一定好，如果日志的流量不是特别大，磁盘性能又跟得上，没有必要一定使用异步日志。

**做人要讲原则，你只要赏我我就喜欢你！**

赞赏支持