Log4j 2.x 日志升级的详细流程（针对各种混合使用Log4j 1.x，Logback，JCL,JUL等）

王寒冰 | 发表于 2017-08-28 | 分类于 日志体系

新的Log4j 2.x版本有了大幅的性能提升、新的插件系统，以及配置设置方面的很多改善。Log4j 1.x 在高并发情况下出现死锁导致cpu使用率异常飙升，而Log4j2.x基于LMAX Disruptor的异步日志在多线程环境下性能会远远优于Log4j 1.x和logback。

一 前言

经过对目前使用比较多的几类日志工具的详细调研、对比和分析（详细的调研结果可以看我的另外一篇博客日志系统的调研与升级），最终得出Log4j 2.x 相对于其他的日志系统有着明显的优势，具体的优势可以用下表进行简单说明：

优化	说明
执行速度	Log4j 2.x 相对于 Log4j 1.x 和 Logback来说，具有更快的执行速度。一方面由于 重写了内部的实现，在某些特定的场景上面，甚至可以比之前的速度快上10倍。比如内部的消息队列采用了ArrayBlockingQueue，相对于原始的ArrayList和锁操作来说，管程类的消息队列拥有更好地性能。同时所需的内存更加少。这是因为Log4j 2.x 采用占位符的形式打印日志（类似于Slf4j门面日志的形式），会先判断一下日志的等级，然后再拼接要打印的内容。另一方面由于Log4j 2.x 充分利用Java 5的并发特性（主要是使用了一些concurrent包下锁），使得性能得到一定的改善，而Log4j 1.x和Logback很多地方还是用的重锁。

优化	说明
异步性能	Asynchronous Loggers是Log4j2新增的日志器，异步日志器在其内部实现采用了LMAX Disruptor（一个无锁的线程间通信库）技术，Disruptor主要通过环形数组结构、元素位置定位和精巧的无锁设计（CAS）实现了在高并发情形下的高性能。而且Log4j 2.x中Asynchronous Appenders作为Asynchronous Loggers工作的一部分，效果进行了增强。每次写入磁盘时，都会进行flush操作，效果和配置“immediateFlush=true”一样。该异步Appender内部采用ArrayBlockingQueue的方式。RandomAccessFileAppender采用ByteBuffer+RandomAccessFile替代了BufferedOutputStream，官方给出的测试数据是它将速度提升了20-200%。
自动加载配置文件	Log4j 2.x 和Logback都新增了自动加载日志配置文件的功能，又与Logback不同，配置发生改变时不会丢失任何日志事件。当Log4j 2.x中配置发生改变时，如果还有日志事件尚未处理，Log4j 2会继续处理，当处理完成后，Logger会重新指向新配置的LoggerConfig对象，并且删除无用的对象。
死锁问题的解决	在Log4j 1.x中同步写日志的时候，在高并发情况下出现死锁导致cpu使用率异常飙升。其中的原因是当一个进程写日志的时候需要获取到Logger和Appender。org.apache.log4j.Logger类继承于org.apache.log4j.Category、Appender继承于org.apache.log4j.AppenderSkeleton。通过Log4j 1.x中Category源码和Appender源码可以知道，当多线程并发时，可能会因为相互持有Logger和Appender发生死锁。 而在log4j 2.x中充分利用Java5的并发支持，并且以最低级别执行锁定。

当然，Log4j 2.x还增加了很多新的特性，想要了解更多的同学，可以前去官网看一下详细的文档http://logging.apache.org/log4j/2.x/。

前面讲了那么多，接下来我们要着重介绍怎么升级到Log4j 2.x, 如果目前的系统是Log1.x、 LogBack、 JCL(Java Commons Logging， 一种门面日志)、JUL(java.util.logging, JDk自带的日志工具)以及前者混合使用的情况，都可以升级到Log4j 2.x(可以不用修改代码，只需要修改pom文件即可)。由于Log4j 1.x、Log4j 2.x 和Logback配置文件的语法相互不兼容，升级的话首先需要修改日志工具的配置文件。下面介绍一下Log4j 2.x 的xml和properties语法（ Log4j 2.x 同样也支持json、yaml类型的配置文件，需要了解的同学可以去官网了解）。

二 配置文件

由于Log4j 2.x 中的语法不兼容 Log4j 1.x 语法，所以为了顺利升级需要将原有的日志配置文件进行转化。Log4j 2.x 和Log4j 1.x 一样会默认从classpath中读取默认的配置文件log4j2.properties或者。log4j2.xml的配置文件。

下面主要简单介绍两种配置文件的语法，详细的配置语法请前往官网（http://logging.apache.org/log4j/2.x/manual/configuration.html）。

当项目启动的时候会默认从classpath加载配置文件，配置文件的名称是log4j2.properties或者log4j2.xml等。如果我们需要指定加载的路径，需要在web.xml中或者程序中加载。注意：虽然spring官方文档里说已经支持Log4j 2.x 配置文件的解析，但是查看高版本的spring，仍然是使用的Log4j 1.x 的解析工具，不知道什么个情况。所以暂时不能用spring对log4j 2.x 配置文件解析。（利用spring加载配置的需要注意了）

2.1 XML配置文件

2.1.1 Configuration标签中的常用元素说明：

monitorInterval：Log4j 2 定期检查和应用配置文件的时间间隔（单位：秒，如果更改配置文件，不用重启系统）。

status：Log4j内部事件记录的等级，只对Log4j本身的事件有效。

strict：是否使用XML Schema来检查配置文件。

schema：Shema文件的位置。

```
<configuration status="warn" monitorInterval="30" strict="true">
    ...
</configuration>
```

2.1.2 Properties 标签说明

标签主要是定义一些常量使用：在下面配置文件中，faerror日志的存储路径是log/faerror.log。

```
<configuration status="warn" monitorInterval="30" strict="true">
    <properties>
        <property name="faerroeFilePath">log/faerror.log</property>
        <property name="crmLogFilePath">log/crmLog.log</property>
    </properties>
    <appenders>
    <RollingFile name="RollingFileError" fileName="${faerrorFilePath}"
                    filePattern="${faerrorFilePath}.%d{yyyy-MM-dd}.txt" append="false">
            <ThresholdFilter level="ERROR"/>
            <PatternLayout pattern="%d - %c [%t] %-5p %c %x %l - %m%n"/>
            <Policies>
                <TimeBasedTriggeringPolicy interval="1" modulate="true"/>
            </Policies>
    </RollingFile>
    ...
</configuration>
```

2.1.3 Appenders标签说明：

Appender：用来定义不同的输出位置，可以是console、文件、远程socket服务器、Apache Flume、JMS以及远程 UNIX系统日志守护进程。一个Logger可以绑定多个不同的Appender。

console: 输出到控制台，target=“SYSTEM_OUT”。

File：会打印日志到文件，这个类型的会每次打印都会清空内容，可以设置属性需不需要清空：append=false。

RollingFile：这个会打印所有的消息，当有策略时，按照策略执行，比如有OnstartupTriggeringPolice的时候，启动就执行TimeBaseTrringPolice 、SizeBasedTriggeringPolice。

fileName: 指定输出日志的目的文件带全路径的文件名.

ThresholdFilter ：指定这个Appender接受、拒绝或者中立的信息级别，当设定好level后，onMatch可以设定为“DENY”、“NEUTRAL”或者“ACCEPT”,同样onMismatc也可以设定以上的三个值。onMatch当设定ACCEPT时，表示符合level设定的日志级别，接受，否则看onMismatch,如果设定的是NEUTRAL,则表示中立，可以将这条信息交给下一个过滤器处理，如果是DENY,则丢弃当前日志。

filePattern: 指定新建日志文件的名称格式.

PatternLayout: 输出格式，不设置默认为:%m%n。

Policies：

TimeBaseTrringPolice：包含两个属性**interval**和**modulate**。基于时间的滚动策略，

interval：integer型，指定两次封存动作之间的时间间隔。单位:以日志的命名精度来确定单位，比如yyyy-MM-dd-HH 单位为小时，yyyy-MM-dd-HH-mm 单位为分钟 。

modulate：boolean型，说明是否对封存时间进行调制。若modulate=true，则封存时将以0点为边界进行偏移计算。比如，modulate=true，interval=4hours，那么
假设上次封存日志的时间为03:00，则下次封存日志的时间为04:00，之后的封存时间依次为08:00，12:00，16:00，。。。

SizeBasedTriggeringPolice ：当日志文件达到指定大小的时候，对日志文件进行拆分，下表是filePattern中的i%，i从0开始；

DefaultRolloverStrategy：按照 filePattern保存的日志文件的数量。如果不做配置，默认是7，这个7指的是上面i的最大值，超过了就会覆盖（开始删除最旧的）之前的日志。

下面是一个完整的日志配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!--设置log4j2的自身log级别为warn-->
<configuration status="warn" monitorInterval="30" strict="true">
    <properties>
        <property name="faerroeFilePath">log/faerror.log</property>
    </properties>
    <appenders>
        <console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="[%d{HH:mm:ss:SSS}] [%p] - %l - %m%n"/>
        </console>
        <File name="log" fileName="log/test.log" append="false">
             <PatternLayout pattern="[%d{HH:mm:ss:SSS}] [%p] - %l - %m%n"/>
        </File>
        <RollingFile name="RollingFileError" fileName="${faerrorFilePath}"
                     filePattern="${faerrorFilePath}.%d{yyyy-MM-dd}.txt" append="false">
            <Filters>
                <!-- 过滤WINFO及以上级别的日志， 如果符合Accept，否则保持中立，交给下一个过滤器处理 -->
```

```
                <ThresholdFilter level="INFO" onMatch="ACCEPT" onMismatch="NEUTRAL"/>
                <ThresholdFilter level="ERROR" onMatch="ACCEPT" onMismatch="DENY"/>
            </Filters>
            <PatternLayout pattern="%d - %c [%t] %-5p %c %x %l - %m%n"/>
            <Policies>
                <OnStartupTriggeringPolicy />
                <TimeBasedTriggeringPolicy interval="1" modulate="true"/>
                <SizeBasedTriggeringPolicy size="100 KB" />
            </Policies>
            <DefaultRolloverStrategy max="20" />
        </RollingFile>
    </appenders>
    <loggers>
        <logger name="org.springframework" level="ERROR"></logger>
        <logger name="es" level="INFO" additivity="false">
            <appender-ref ref="RollingFileES"/>
        </logger>
        <root level="info" includeLocation="true">
            <appender-ref ref="Console"/>
            <appender-ref ref="RollingFileError"/>
        </root>
    </loggers>

</configuration>
```

2.1.4 Loggers标签说明：

它被用来配置LoggerConfig，包含一个root logger和若干个普通logger,普通必须有一个name元素，root logger不用name元素。

每个logger可以指定一个level（TRACE, DEBUG, INFO, WARN, ERROR, ALL or OFF），不指定时level默认为ERROR。

additivity指定是否同时输出log到父类的appender，缺省为true。（层级关系与Java包类似，例如：com.foo是com.foo.Bar的父级；java是java.util的父级，是java.util.vector的祖先。root是最顶层。）

每个logger可以包含若干个属性：AppenderRef, Filter, Layout, 等。

```
<loggers>
    <logger name="org.springframework" level="ERROR"></logger>
    <logger name="es" level="INFO" additivity="false">
        <appender-ref ref="RollingFileES"/>
    </logger>
    <root level="info" includeLocation="true">
```

```
        <appender-ref ref="Console"/>
        <appender-ref ref="RollingFileError"/>
    </root>
</loggers>
```

2.2 properties 配置文件

properties配置文件的语法大致和xml语法相同，只是写法格式不同而已。其中crmLog是我们自定义的Logger。

在书写Log4j 2.x 配置文件的时候，我们应该首先配置Appender，Appender表示往哪个输出源进行输入。然后在配置logger,logger代表的是日志实例。logger里面有对appender的引用，表示这个logeer可以往哪个输出源进行输入。具体的流程如下：

```
graph LR
定义Appender-->定义Logger
```

像对appender和logger组件进行定义的时候，appender.XXX....和logger.YYY...的,其中XXX 和YYY 表示组件的别称，只是为了区分不同的组件，具有唯一性，可以任意命名，不过强烈建议以你的logger的name进行命名，一方面能够更加直观，另一方面也能保持唯一性（因为每个Logger的定义只能有一个）。第三方jar的别名可以考虑把点去掉，如logger.orgspringframework.name = org.springframework，其中orgspringframework只是把logger的名称中的org.springframework的.去掉作为别名，容易理解。

```
status = info
name = PropertiesConfig

property.faerrorFileName = log/faerror.log
property.crmLogFileName = log/crmLog.log

appender.console.type = Console
appender.console.name = console
appender.console.layout.type = PatternLayout
appender.console.layout.pattern = %d - %c [%t] %-5p %c %x %l - %m%n

appender.error.type = RollingFile
appender.error.name = error
appender.error.filter.threshold.type = ThresholdFilter
appender.error.filter.threshold.level = warn
appender.error.fileName = ${faerrorFileName}
appender.error.filePattern = ${faerrorFileName}.%d{yyyy-MM-dd}
appender.error.layout.type = PatternLayout
appender.error.layout.pattern = %d - %c [%t] %-5p %c %x %l - %m%n
appender.error.policies.type = Policies
```

```
appender.error.policies.time.type = TimeBasedTriggeringPolicy
appender.error.policies.time.interval = 1
appender.error.policies.time.modulate = true

appender.crmLog.type = RollingFile
appender.crmLog.name = crmLog
appender.crmLog.fileName = ${crmLogFileName}
appender.crmLog.filePattern = ${crmLogFileName}.%d{yyyy-MM-dd}
appender.crmLog.layout.type = PatternLayout
appender.crmLog.layout.pattern = %d - %c [%t] %-5p %c %x %l - %m%n
appender.crmLog.policies.type = Policies
appender.crmLog.policies.time.type = TimeBasedTriggeringPolicy
appender.crmLog.policies.time.interval = 1
appender.crmLog.policies.time.modulate = true

logger.crmLog.name = crmLog
logger.crmLog.level = info
logger.crmLog.additivity = false
logger.crmLog.appenderRef.crmlog.ref = crmLog
logger.crmLog.appenderRef.error.ref = error

logger.orgspringframework.name = org.springframework
logger.orgspringframework.level = info
logger.orgapachestruts2.name = org.apache.struts2
logger.orgapachestruts2.level = info
logger.comopensymphony.name = com.opensymphony
logger.comopensymphony.level = info
logger.comibatis.name = com.ibatis
logger.comibatis.level = info

rootLogger.level = info
rootLogger.appenderRef.console.ref = console
rootLogger.appenderRef.error.ref = error
```

特别注意：logger.crmLog.appenderRef.crmlog.ref = crmLog中appenderRef后面跟的是Appender的名称的小写，比如crmLog这个名称中有大写字母，跟在appenderRef后面需要全小写。

三 桥接器和适配器的介绍

项目中升级到Log4j 2.x ,普遍遇到的问题是如果当前环境使用的Log4j 1.x（甚至是直接调用Log4j 1.x的接口）、logback、jcl（Apache Commons Logging）和jul（java.util.logging）混合使用的情况下，如何做到版本的稳定升级而不会影响现有项目的业务或者代码的大量修改。桥接器很好的完成了旧版本到新版本的应用。以下对可能使用的桥接器、适配器及使用到的Log4j相关的jar包进行简单介绍：

Jar包名称	作用
slf4j-api	即简单日志门面（Simple Logging Facade for Java）。从设计模式的角度考虑，它是用来在log和代码层之间起到门面的作用。对用户来说只要使用slf4j提供的接口，即可隐藏日志的具体实现。这与jdbc和相似。使用jdbc也就避免了不同的具体数据库。使用了slf4j可以对客户端应用解耦。因为当我们在代码实现中引入log日志的时候，用的是接口，所以可以实时的更具情况来调换具体的日志实现类。这就是slf4j的作用。
log4jdbc	对sql语句执行的详细信息日志进行打印。
log4j-web	用来加载Log4j 2.x 的配置文件，以及动态监测并加载Log4j 2.x 的配置。
log4j-slf4j-impl	用来绑定门面日志slf4j和日志工具Log4j 2.x 的适配器
log4j-core	log4j 2.x 日志工具的实现类
log4j-api	log4j 2.x 日志工具的API
jul-to-slf4j	如果程序中原来使用了java.util.logging，把日志重定向到slf4j门面日志。
jcl-over-slf4j	如果程序中原来使用了Apache Commons Logging，把日志重定向到slf4j门面日志。
disruptor	Log4j 2.x 中如果使用了异步日志的功能，这个jar包是必须引入的，用于异步并发的高性能通信框架。
log4j-1.2-api	由于Log4j 1.x 和Log4j 2.x 有些接口不兼容，项目的代码中有直接调用Log4j 1.x的接口。为了不用修改已有的代码，可以加上这个jar实现。

四 (Log4j 1.x、Logback、JUL、JCL)升级为 Log4j 2.x

4.1 pom文件的修改

Log4j 1.x 主要依赖的jar包如下，其中slf4j-api是日志的门面日志，slf4j-log4j12是实现log4j 1.x到门面日志的绑定，log4j是Log4j日志系统的实现类。

4.1.1 去掉pom文件中Log4j 1.x 的相关jar包(如果项目里面包含Log4j 1.x).

```
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>${slf4j.version}</version>
        <scope>runtime</scope>
</dependency>
<dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
</dependency>
```

4.1.2 去掉pom文件中Logback 的相关jar包(如果项目里面包含logback)

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-core</artifactId>
    <version>0.9.28</version>
    <type>jar</type>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>0.9.28</version>
    <type>jar</type>
</dependency>
```

4.1.3 排除Log4j 1.x 的依赖(如果里面包含Log4j 1.x)

需要确定项目pom文件中依赖的其他的jar中也不再依赖log4j及slf4j-log4j12，具体方式可以通过IDE提供的功能或者直接使用mvn dependency:tree确定依赖关系。

```
<exclusions>
    <exclusion>
```

```
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
    </exclusion>
    <exclusion>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
    </exclusion>
</exclusions>
```

例如一下的排除示例：

```
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
    <version>3.4.6</version>
    <exclusions>
        <exclusion>
                <groupId>org.slf4j</groupId>
                <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
        <exclusion>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

4.1.4 添加Log4j 2.x 的依赖

```
<!-- LOGGING -->
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.6</version>
</dependency>
                <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.8.2</version>
</dependency>
<dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.8.2</version>
```

```
</dependency>
<dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-web</artifactId>
        <version>2.8.2</version>
</dependency>
<dependency> <!-- 桥接：告诉Slf4j使用Log4j2 -->
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.8.2</version>
</dependency>
<dependency>
    <groupId>com.lmax</groupId>
    <artifactId>disruptor</artifactId>
    <version>3.3.4</version>
</dependency>
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
        <version>1.7.6</version>
        <scope>runtime</scope>
</dependency>
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jul-to-slf4j</artifactId>
        <version>1.7.6</version>
        <scope>runtime</scope>
</dependency>
<dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-1.2-api</artifactId>
        <version>2.8.2</version>
</dependency>
```

log4j-api和log4j-core是Log4j 2.x的核心jar包，不可缺少。log4j-slf4j-impl是实现log4j 2.x 和slf4j 门面日志的绑定，至于disruptor是实现异步日志的并发框架，

/**注意*/**

有些第三方库中利用了slf4j-log4j12 jar包去打印日志，需要手动排除。如果不排除的话会报以下错误

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/C:/Users/bjwanghanbing/.m2/repository/org/slf4j/slf4j-log4j12/1.7.6/slf4j-log
SLF4J: Found binding in [jar:file:/C:/Users/bjwanghanbing/.m2/repository/org/apache/logging/log4j/log4j-slf4j-im
```

```
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

这是因为动态绑定的时候桥接器重复，程序不知道使用哪一个，所以需要排除项目中的slf4j-log4j12。

五 配置文件加载

Log4j-web是实现监控Log4j 2.x 配置文件，Log4j 2.x中有一个新的特点是定时检测并加载配置文件。web.xml中配置如下图所示：

```
<context-param>
        <param-name>log4jConfiguration</param-name>
        <param-value>classloader:/config/${spring.profiles.active}/log4j2.properties</param-value>
</context-param>
<listener>
        <listener-class>org.apache.logging.log4j.web.Log4jServletContextListener</listener-class>
</listener>
```

注意：一定用classloader，不可以用classpath。

其中spring.profiles.active是启动时配置的环境变量，这个一定要设置。

六 JUL 日志的重定向

（以下这一步不是必须的，除非原先使用的JUL，才需要下面的步骤）

注意：如果项目中或者第三方jar包中用到了jdk自带的日志工具（java.util.logging.Logger）需要在项目的启动过程中进行重定向。虽然pom文件里面加入了jul-to-slf4j作为桥接器，把jul的日志流量桥接到slf4j门面日志。但是JUL仍然需要在代码层面重新定向一下，目前有两种方式.

一种是编程实现:

当我们进行单元测试的时候，需要在测试环境启动之前编码进行重定向，如下代码所示：

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "/application-context.xml" })
public class TestLog {

        Logger log = Logger.getLogger("testLog");

        @BeforeClass
        public static  void startupInit() {

```

```
10                  //System.setProperty("spring.profiles.active", "dev");
11                  //手动加载Log4j 2.x 的配置文件
12                  LoggerContext context = (LoggerContext) LogManager.getContext(false);
13                  context.setConfigLocation(URI.create("/"+System.getProperty("user.dir").replaceAll("\\\\", "/")
14                  context.reconfigure();
15
16                  //用于重定向JUL。
17                  SLF4JBridgeHandler.removeHandlersForRootLogger();
18                  SLF4JBridgeHandler.install();
19          }
20
21          @Test
22          public void testLog() {
23                  log.info("this is a test Log");
24          }
25
26  }
```

另一种是扩展Spring提供的org.springframework.web.util.Log4jConfigListener，在TOMCAT启动的时候编程方式设置JDK的日志只是用SLF4JBridgeHandler进行处理。

如下代码所示:

```
1   public class Log4jReConfigListener extends Log4jServletContextListener {
2
3           @Override
4       public void contextInitialized (ServletContextEvent event) {
5           instalJulToSlf4jBridge();
6           super.contextInitialized(event);
7       }
8
9       public void instalJulToSlf4jBridge() {
10          SLF4JBridgeHandler.removeHandlersForRootLogger();
11          SLF4JBridgeHandler.install();
12      }
13  }
```

然后web.xml中：（如果上面的步骤中已经添加不需要加了）

```
1   <listener>
2           <listener-class>xxx.yyyy.Log4jReConfigListener</listener-class>
3   </listener>
```

至此，所有的升级完成，升级完成后，注意观察打出的日志是否正常，

性能优化 Log4j SLF4J 死锁 logback 异步通信

Spring-SpringMVC父子容器&AOP使用总结

SQL vs NoSQL —Mysql和MongoDB性能对比及应用场景分析