日志规范

V1.0

**京东生活旅行**

**版本修改记录**

| **版本号** | **版本修改说明** | **作者/日期** | **审批人/日期** | **生效日期** |
| --- | --- | --- | --- | --- |
| 1.0 |  | 架构师/20190226 | 架构师/20190306 | 201920306 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

说明： C：Create，初始创建；A：Add，增加内容；M：Mod，修改；D：Del，删除

目 录

[1. 概述 4](#_Toc2846807)

[1.1. 目的 4](#_Toc2846808)

[1.2. 适用范围 4](#_Toc2846809)

[2. 日志记录准则 4](#_Toc2846810)

[3. 日志框架 4](#_Toc2846811)

[4. 日志级别 5](#_Toc2846812)

[5. 记录时机 5](#_Toc2846813)

[6. 配置规范 6](#_Toc2846814)

[6.1. logback配置参考 6](#_Toc2846815)

[6.2. Log4j2配置参考 10](#_Toc2846816)

[7. 日志使用规约 10](#_Toc2846817)

## 概述

### 目的

为了规范应用日志，方便系统问题分析跟查，特制订本规范。

### 适用范围

生旅研发所有后台应用系统。

## 日志记录准则

记录系统运行异常信息

记录系统运行状态信息

记录系统运行性能指标

## 日志框架

新系统：推荐使用slf4j + log4j2 + lombok简化日志处理，作为日志框架。

遗留系统：持续使用slf4j+logback+lombok简化日志处理，作为日志框架。

SLF4J是一个用于日志系统的简单[Facade](https://baike.baidu.com/item/Facade" \t "_blank)，logback是slf4j的具体实现。

Logback性能较好，日志展现形式较美观，完整实现了SLF4J API。

Log4j2是重新架构、抛弃了之前log4j的不足，吸取logback的设计重新推出的一款新组件，异步记录日志性能提升尤其明显。

使用lombok简化了日志组件的使用，利用@Slf4j进行日志注解，使用该注解会创建一个log属性，可以直接log.info()、log.error()。

## 日志级别

* ERROR：系统发生了错误事件，但仍然不影响系统的继续运行。系统需要将错误或异常细节记录ERROR日志中，方便后续人工回溯解决。
* WARN：系统在业务处理时触发了异常流程(参数验证不过)，但系统可恢复到正常态，下一次业务可以正常执行。如程序调用了一个旧版本的接口，可选参数不合法，非业务预期的状态但仍可继续处理等
* INFO：记录系统关键信息，旨在保留系统正常工作期间关键运行指标，开发人员可以将初始化系统配置、业务状态变化信息，或者用户业务流程中的核心处理记录到INFO日志中，方便日常运维工作以及错误回溯时上下文场景复现
* DEBUG：可以将各类详细信息记录到DEBUG里，起到调试的作用，包括参数信息，调试细节信息，返回值信息等。

## 记录时机

* **系统初始化：**系统初始化时会依赖一些关键配置，根据参数不同会提供不一样的服务。将系统的启动参数记录INFO日志，打印出参数以及服务启动完成状态。
* **业务流程与预期不符：**系统中结果与期望不符，应当记录日志。常见的合适场景包括外部参数不正确，数据处理问题导致返回码不在合理范围内等等。
* **系统核心的关键动作：**系统中核心角色触发的业务动作是需要多加关注的，是衡量系统正常运行的重要指标，建议记录INFO级别日志，比如微服务各服务节点交互等。
* **捕获到异常时**：这类捕获的异常是系统告知开发人员需要加以关注的，应当记录日志，根据实际情况使用warn或者error级别。
* **外部接口日志：**这类日志涉及到与外部系统的交互，事关责任问题，建议将原始数据文件内容写入日志或数据库（如mongodb），核心处理逻辑关键业务数据也尽量写入日志。如果涉及到重发，建议将处理失败的原始数据文件日志写入数据库，以便重发执行。

## 配置规范

### logback配置参考

统一使用logback.xml配置，logback.xml 文件放在 classpath 目录下。

所有的jar包中不建议包含logback.xml文件，避免干扰实际的业务系统。

通过在文件logback.xml中引入资源文件log.properties定义logback属性信息，log.properties根据不同的profile放置在不同位置；

|  |
| --- |
| <property resource="log.properties"/> |

**log.properties文件**

属性命名推荐使用统一使用大写，以下划线分隔，参考

|  |
| --- |
| APP\_NAME = yourAppName  LOG\_DIR = /export/home/logs/yourSystem/yourAppName  LOG\_PATTERN = [%date{yyyy-MM-dd HH:mm:ss.SSS}] %level [%mdc{invokeNo}] %C{0}:%line - %message%n |

注意Logger间的继承关系，推荐additivity设置false：

* 子logger会默认继承父logger的appender，将它们加入到自己的Appender中；除非加上了additivity="false"，则不再继承父logger的appender。
* 子logger只在自己未定义输出级别的情况下，才会继承父logger的输出级别。

将日志输出到文件当中，禁止使用FileAppender，推荐使用提供自动切换功能的RollingFileAppender

Log文件位置和命名，目前Log文件的位置统一放在相同目录下面。

|  |  |
| --- | --- |
| 文件名 | 描述 |
| /export/home/logs | 默认日志路径（所有日志的根路径） |
| /export/home/logs/${SYSTEM\_NAME}/${APP\_NAME} | log.properties中配置的日志全路径LOG\_DIR |
| ${LOG\_DIR}/all.log | 必选 |
| ${LOG\_DIR}/ all-%d{yyyy-MM-dd}.log | All历史文件命名 |
| ${LOG\_DIR}/all\_error.log | 必选 |
| ${LOG\_DIR}/sql.log | 可选 |

日志按天记录，单个日志文件最大不超过2000MB，考虑到有些bug按月规律出现，推荐历史日志保留30天。

|  |
| --- |
| <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  <fileNamePattern>${LOG\_DIR}/all-%d{yyyy-MM-dd}.log</fileNamePattern>  <maxHistory>30</maxHistory>  <timeBasedFileNamingAndTriggeringPolicy  class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  <!-- 每个日志文件大小不超过2GB -->  <maxFileSize>2000MB</maxFileSize>  </timeBasedFileNamingAndTriggeringPolicy>  </rollingPolicy> |

在出现问题之后，需要立即根据日志定位问题。对于INFO及以上级别的日志，要求按照一定顺序，输出以下必要的信息。参考日志格式定义

|  |
| --- |
| <encoder charset="UTF-8">  <pattern>[%date{yyyy-MM-dd HH:mm:ss.SSS}] %level [%mdc{invokeNo}] %C{0}:%line - %message%n</pattern>  </encoder> |

一个完整的Appender配置如下

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <configuration>  <property resource="log.properties"/>  <contextName>${APP\_NAME}</contextName>  <appender name="console" class="ch.qos.logback.core.ConsoleAppender">  <encoder charset="UTF-8">  <pattern>${LOG\_PATTERN}</pattern>  </encoder>  </appender>  <appender name="all" class="ch.qos.logback.core.rolling.RollingFileAppender">  <file>${LOG\_DIR}/all.log</file>  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  <!-- 日志每天进行rotate -->  <fileNamePattern>${LOG\_DIR}/all-%d{yyyy-MM-dd}.%i.log</fileNamePattern>  <maxHistory>30</maxHistory>  <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  <!-- 每个日志文件大小不超过2GB -->  <maxFileSize>2000MB</maxFileSize>  </timeBasedFileNamingAndTriggeringPolicy>  </rollingPolicy>  <encoder charset="UTF-8">  <pattern>${LOG\_PATTERN}</pattern>  </encoder>  <filter class="ch.qos.logback.classic.filter.ThresholdFilter">  <level>info</level>  </filter>  </appender>  <appender name="all-error" class="ch.qos.logback.core.rolling.RollingFileAppender">  <file>${LOG\_DIR}/all-error.log</file>  <filter class="ch.qos.logback.classic.filter.LevelFilter">  <level>ERROR</level>  <onMatch>ACCEPT</onMatch>  <onMismatch>DENY</onMismatch>  </filter>  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  <fileNamePattern>${LOG\_DIR}/all-error-%d{yyyy-MM-dd}.%i.log</fileNamePattern>  <maxHistory>30</maxHistory>  <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  <!-- 每个日志文件大小不超过2GB -->  <maxFileSize>2000MB</maxFileSize>  </timeBasedFileNamingAndTriggeringPolicy>  </rollingPolicy>  <encoder charset="UTF-8">  <pattern>[%date{yyyy-MM-dd HH:mm:ss.SSS}] %level [%mdc{invokeNo}] %C{0}:%line - %message%n</pattern>  </encoder>  </appender>  <root level="debug">  <appender-ref ref="all"/>  <appender-ref ref="all-error"/>  <appender-ref ref="console"/>  </root>  </configuration> |

### Log4j2配置参考

统一使用log4j2.xml配置，log4j2.xml 文件放在 resource目录下。

注意Logger间的继承关系，推荐additivity设置false：

* 子logger会默认继承父logger的appender，将它们加入到自己的Appender中；除非加上了additivity="false"，则不再继承父logger的appender。
* 子logger只在自己未定义输出级别的情况下，才会继承父logger的输出级别。

将日志输出到文件当中，考虑到RollingRandomAccessFile比RollingFile更灵活，推荐统一使用RollingRandomAccessFile。

Log文件位置和命名，目前Log文件的位置统一放在相同目录下面。

|  |  |
| --- | --- |
| 文件名 | 描述 |
| /export/home/logs | 默认日志路径（所有日志的根路径） |
| /export/home/logs/${SYSTEM\_NAME}/${APP\_NAME} | log.properties中配置的日志全路径LOG\_DIR |
| ${LOG\_DIR}/all.log | 必选 |
| ${LOG\_DIR}/ all-%d{yyyy-MM-dd}.log | All历史文件命名 |
| ${LOG\_DIR}/all\_error.log | 必选 |
| ${LOG\_DIR}/sql.log | 可选 |

日志按天记录，单个日志文件最大不超过3000MB，考虑到有些bug按周规律出现，推荐历史日志保留14天

|  |
| --- |
| <RollingRandomAccessFile name="all-append" immediateFlush="true" fileName="${LOG\_DIR}/all.log" filePattern="${LOG\_DIR}/all-%d{yyyy-MM-dd}-%i.log">  <Policies>  <SizeBasedTriggeringPolicy size="3GB" />  <TimeBasedTriggeringPolicy interval="8" modulate="true" />  <!-- 最多备份14天以内||日志文件大小达到50GB的日志||文件数量超过20此处为策略限制，Delete中可以按自己需要用正则表达式编写 -->  <DefaultRolloverStrategy>  <Delete basePath="${filePath}" maxDepth="1">  <IfLastModified age="14d" />  <IfAccumulatedFileSize exceeds="50 GB" />  <IfAccumulatedFileCount exceeds="20" />  </Delete>  </DefaultRolloverStrategy> |

在出现问题之后，需要立即根据日志定位问题。对于INFO及以上级别的日志，要求按照一定顺序，输出以下必要的信息。参考日志格式定义

|  |
| --- |
| <PatternLayout>  <Pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level [%mdc{invokeNo}] %C{2}:%L %M - %msg%n</Pattern>  </PatternLayout> |

一个完整的Appender配置如下

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!--  Configuration后面的status，这个用于设置log4j2自身内部的信息输出，可以不设置，当设置成trace时，你会看到log4j2内部各种详细输出。  monitorInterval：Log4j能够自动检测修改配置 文件和重新配置本身，设置间隔秒数。  -->  <configuration status="OFF" monitorInterval=”600″>  <properties>  <property name="LOG\_PATH">/export/home/logs/yourSystem/yourAppName</property>  <property name="LOG\_PATTERN">%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level [%mdc{invokeNo}] %C{2}:%L %M - %msg%n </property>  <property name="EVERY\_FILE\_SIZE">3GB</property>  <property name="OUTPUT\_LOG\_LEVEL">info</property>  <property name="FILE\_COUNT">20</property>  <property name="ERROR\_FILE\_COUNT">3</property>  </properties>    <!--先定义所有的appender-->  <appenders>  <!--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="info" onMatch="ACCEPT" onMismatch="DENY"/>  <!--输出日志的格式-->  <PatternLayout pattern="${LOG\_PATTERN}"/>  </Console>  <!-- 打印信息，每次大小超过size，则这size大小的日志会自动存入按年份-月份建立的文件夹下面并进行压缩，作为存档-->  <RollingRandomAccessFile  name="all"  immediateFlush="true"  fileName="${LOG\_DIR}/all.log"  filePattern="${LOG\_DIR}/all-%d{yyyy-MM-dd}-%i.log">  <Filters>  <ThresholdFilter level="error" onMatch="ACCEPT" onMismatch="NEUTRAL"/>  <ThresholdFilter level="warn" onMatch="ACCEPT" onMismatch="NEUTRAL"/>  <ThresholdFilter level="info" onMatch="ACCEPT" onMismatch="NEUTRAL"/>  <ThresholdFilter level="trace" onMatch="DENY" onMismatch="DENY"/>  </Filters>  <PatternLayout>  <Pattern>${LOG\_PATTERN}</Pattern>  </PatternLayout>  <Policies>  <SizeBasedTriggeringPolicy size="${EVERY\_FILE\_SIZE}" />  <TimeBasedTriggeringPolicy interval="1" modulate="true" />  </Policies>  <!-- DefaultRolloverStrategy属性如不设置，则默认为最多同一文件夹下7个文件，这里设置了20 -->  <DefaultRolloverStrategy>  <!-- 最多备14 天以内||日志文件大小达到50GB的日志||文件数量超过20此处为策略限制，Delete中可以按自己需要用正则表达式编写 -->  <Delete basePath="${LOG\_DIR}" maxDepth="1">  <IfLastModified age="14d" />  <IfAccumulatedFileSize exceeds="50 GB" />  <IfAccumulatedFileCount exceeds="20" />  </Delete>  </DefaultRolloverStrategy>格  </RollingRandomAccessFile>    <RollingRandomAccessFile  name="all-error"  immediateFlush="true"  fileName="${LOG\_DIR}/all-error.log"  filePattern="${LOG\_DIR}/all-error-%d{yyyy-MM-dd}-%i.log">  <Filters>  <ThresholdFilter level="error" onMatch="ACCEPT" onMismatch="NEUTRAL"/>  <ThresholdFilter level="warn" onMatch="ACCEPT" onMismatch="DENY"/>  </Filters>  <PatternLayout>  <Pattern>${LOG\_PATTERN}</Pattern>  </PatternLayout>  <Policies>  <SizeBasedTriggeringPolicy size="${EVERY\_FILE\_SIZE}" />  <TimeBasedTriggeringPolicy interval="1" modulate="true" />  </Policies>  <!-- DefaultRolloverStrategy属性如不设置，则默认为最多同一文件夹下7个文件，这里设置了20 -->  <DefaultRolloverStrategy max="${ERROR\_FILE\_COUNT}"/>  </RollingRandomAccessFile>  </appenders>  <!--然后定义logger，只有定义了logger并引入的appender，appender才会生效-->  <loggers>  <!--默认的root的logger-->  <Logger name="all" level="info" additivity="false">  <AppenderRef ref="all" />  </Logger>  <Logger name="all-error" level="error" additivity="false">  <AppenderRef ref="all-error" />  </Logger>  <root level="${OUTPUT\_LOG\_LEVEL}">  <appender-ref ref="console"/>  </root>  </loggers>  </configuration> |

## 日志使用规约

1. 使用@SLF4J中的API进行日志打印。
2. 日志输出必须采用UTF-8字符集，推荐打印日志时输出英文，防止中文不支持而打印出乱码的情况。
3. 不允许记录日志后又抛出异常，因为这样会多次记录日志，只允许记录一次日志，应抛出异常，顶层打印一次日志。

|  |
| --- |
| try {  //错误  } catch (Exception e) {  log.error("xxxxxx", e);  throw e；  } |

1. 输出Exceptions的全部堆栈信息。

|  |
| --- |
| //错误例子, 丢失掉StackTrace信息  log.error(e.getMessage());  //错误例子,丢失掉StackTrace信息  log.error(“Bad things : {}“, e.getMessage());  //正确例子  log.error(“Bad things : {}“,e); |

1. 禁止system.out 用于日志记录。
2. 线上必须关闭 DEBUG 级别日志。
3. 非正常的情况，需要根据情况选择打印warn 或 error 日志。
4. 日志输出，必须使用占位符的方式，因为即使信息不打印，也会执行字符串拼接，造成资源浪费。
5. 日志中不允许出现计算或方法调用，防止在打印日志的时候报错。
6. 输出的POJO类必须重写toString方法，否则只输出对象的hashCode值，没有参考意义。
7. 不记录对于排查故障毫无意义的日志信息，日志信息一定要带有业务信息

|  |
| --- |
| //错误  log.error(“handle failed“);  //正确  log.error(“handle failed，id= {}“, id); |

1. 禁止大量无效重复的日志输出，即通常情况下在程序日志里只记录一些比较有意义的状态数据，参考日志记录时机。
2. 不可以讲敏感业务信息记录入日志文件
3. 严防日志占满磁盘，定期检查磁盘（确定是否有磁盘告警）。