<https://mp.weixin.qq.com/s/KcmRaV628CwvDlGGbPOXRQ>

日志规范多重要，这篇文章告诉你！

## 本文脉络

* 1 日志的重要性
  + 1.1 写好业务代码很重要
  + 1.2 记好日志一样重要
* 日志要素
* 记录时机
* 日志框架
* 总结

## 1 日志的重要性

### 1.1 写好业务代码很重要

在开发过程中，写好业务代码的重要性不言而喻，毕竟这是跟程序功能最密切相关的工作，程序出 BUG，一般就是业务代码写得有问题了。

### 1.2 记好日志一样重要

做好日志记录，也是非常重要的，通过查看日志，能帮我们解决很多问题，以下是开发过程中经常会碰到的一些问题：

**程序是不是按预期执行？**

我们可以通过日志记录程序的执行流程、运行状态、关键指标⋯⋯有了这些日志，才能更好地进行调试跟踪。

**程序有BUG怎么办？**

有BUG得排查问题，而定位问题最高效的方式，就是日志。我们总不能一行行代码排查，或者掐指一算随便蒙，不然问题没找到，更不用谈修复上线。

**用户在系统上干了什么？**

这就需要结合业务，记录用户操作行为。用户登录到退出系统中间，所有的重要操作，都应该形成日志，作为审计的依据。

**这个问题是谁造成的？**

在大型应用中，不同系统之间的协作相当紧密，有时系统出问题了，可能是第三方系统造成的，通过在程序交互的关键位置记好日志，就能减少不必要的扯皮。

## 日志要素

既然日志这么重要，一份好的日志，应该具备哪些要素呢？结合工作中的积累，我把日志要素总结为以下这张图（绿色部分是必选项，红色部分酌情选择）：

**日志时间**

日志作为事件的表述，事件发生的时间一定要有，而且应该精确，推荐的格式是:

yyyy-MM-dd HH:mm:ss.SSS

**日志级别**

应该根据日志的重要性或严重程度划分等级，最常用的日志级别有：DEBUG、INFO、WARN、ERROR，只有合理定义日志级别，才能避免日志混乱。

* DEBUG：调试，在开发、测试阶段使用，记录调试性质的内容，生产上应该关闭该级别的日志，减少频繁打印导致程序性能受影响及存储空间的浪费。
* INFO：信息，记录系统正常运行期间的关键信息，主要是程序运行度量、业务操作记录两部分内容，分别用于系统监控及业务审计。
* WARN：警告，用于程序告警，造成这些告警的原因是可预期有规划的，比如请求参数不合法、系统资源紧张但可等待恢复、依赖方不可用但允许临时降级。
* ERROR：错误，记录不可预知的程序异常。比如网络调用、数据库访问等低层资源造成的错误，应用层面没有预期到的，这种错误日志数量上应该很少，内容上应该尽量详尽记录出错成因。

**线程名称**

特别在 Web 应用程序中，一次同步请求一般对应一个处理线程，输出线程名称可以区分一次具体的请求上下文。

**业务标识**

用来区分日志属于哪块业务，因为日志都是跟业务相关联的，通过该部分内容便于按业务进行日志归类聚合。

**记录器名称**

日志的记录器名称一般是声明日志记录器实例的类名，通过记录器名称可以快速定位到日志输出的类是哪个。

**日志内容**

根据不同的日志等级，在日志内容上会有不同的侧重点，但在内容输出上，有一些点是需要注意的：

* 通过日志框架而不是直接使用程序输出流（System.out、System.err）来打印日志内容。
* 除了开发环境，其它线上环境应该采用文件日志而不是控制台日志，一方面是日志持久化，另一方面是程序性能考虑。
* 日志内容应该异步输出，而不是同步阻塞的方式，特别是大迸发应用场景下更应该注意。
* 善用占位符特性，一般日志框架都有占位符功能，避免不必要的字符串拼接。
* 预防日志内容构建过程中的空指针异常、多余的对象序列化、不必要的中间过程执行。
* 注意敏感内容输出，造成的安全隐患，比如用户密码、账户信息等。

**OpenTracing 标识**

通过 AOP 切面，日志框架 MDC 等技术结合，在日志上添加一些链路追踪的扩展元素，将会很大程度方便通过日志进行程序请求调用的链路追踪，这在分布式系统中尤其重要。

**异常堆栈**

堆栈异常信息有助于程序异常的排查定位，但这部分信息的记录输出，对系统性能有一定的影响，应该酌情考虑，如果记录异常信息足够定位异常的，就不要打印整个堆栈。一般是ERROR打印异常堆栈，WARN不打印，还有就是通过日志框架打印，避免用printStackTrace方法打印。

**产生行数**

即产生日志的源代码行，该记录对程序性能有比较大的影响，也应该谨慎使用，前面谈到通过记录器名称跟类名关联，很大程度上能帮我们定位产生日志的代码范围，行信息就可以不用了。



## 记录时机

**程序流程**

记录程序的流转分支，在关键代码逻辑的执行前后进行相应的日志输出，有助于代码调试。但要避免不必要的日志输出，比如一般只在循环体前后记录日志，而不在循环体内重复记录，过多的日志反而会影响阅读。

**远程调用**

远程调用也属于程序流程的一部分，但第三方远程调用的日志信息级别，应该比一般的调试日志区别对待，因为涉及到外部系统的交互，在出入口处记录请求响应的信息，相当重要。

**系统初始化**

系统初始化需要依赖一些关键配置参数，这些参数决定系统的启动状态，应该把这些系统初始化信息记录起来。

**核心业务操作**

系统用户进行核心业务操作的行为，也应该进行记录，便于进行操作审计。

**可预期的异常**

这类异常应该有效记录起来，通过警告方式反馈给相关人员加以关注，避免频繁发生，最终演化为不可控的错误。

**预期外的错误**

这类异常发生时，要有详尽记录，并通知相关人员介入处理，第一时间作出响应，因为这种错误已经影响系统的正常使用。

## 日志框架

在 Java 应用中，日志打印涉及到的组件工具主要如下图所示：



中间部分是日志输出的核心，除了 Logback 外，常用的还有 Log4j/Log4j 2，Log4j 已经有点过时，而 Log4j 2 在异步日志的性能表现上，比 Logback 更加突出。

## 总结

本文主要讲的是日志的一些相关概念，包括日志要素、记录时机、日志框架的简单介绍，日志很重要，好的系统离不开日志。后续会对这些进行细化，从技术实现的角度，分享应用日志性能提升、日志采集汇总、日志统计分析、分布式系统链路跟踪等内容，以及在工作中遇到的一些日志坑，后续会分享架构其他方向文章。