**日志方案**

**处理原则**

通常来说日志分为两种：业务日志和异常日志，使用日志我们希望能达到以下目标：

1. 对程序运行情况的记录和监控

2. 在必要时可详细了解程序内部的运行状态

3. 对系统性能的影响尽量小

**配置策略**

存放路径：${catalina.base}/logs

日志模板：[INFO]-[2018-03-24 18:34:35]-[类名.函数名():行]-[线程名称] 信息详情

系统日志按级别分为两个：

error.log，ERROR级别

output.log，INFO级别，包含error.log中的日志

单个日志文件最大50MB，超出生成xxx.log1、xxx.log2，最多保留10个文件

模块日志（可选）：

对一些需要隔离的模块日志，可根据情况另行配置

第三方库日志：设置为ERROR级别，包括springframework、ehcache、druid等

**编码规范**

1. 使用 slf4j，不要直接使用日志实现框架的API（Log4j、Logback等）

2. 一个对象中只应有一个日志变量，定义为final static，变量名全大写，例如：

private static final Logger LOG = LoggerFactory.getLogger(当前类.class);

3. 合理使用日志级别：

fatal - 造成服务中断的致命错误

error - 其他错误运行期错误

warn - 警告信息，如程序调用了一个即将作废的接口，接口的不当使用，运行状态不是期望的但仍可继续处理等

info - 有意义的事件信息，如程序启动事件、运行消耗时间等

debug - 调试信息，可记录详细的业务处理到哪一步了，以及当前的变量状态

trace - 更详细的跟踪信息

4. 正式代码中不允许出现 System.out、e.printStackTrace 等语句（单元测试除外）

5. 对于error和fatal级别，一定要写明详细的原因，并带上调用堆栈（如果有）：

catch (IOException e) {

e.printStackTrace(); // 错误，参考上一条规则

LOG.error("xx"); // 错误

LOG.error("xx", e.getMessage()); // 错误

LOG.error("xx", e); // 正确

}

6. 不允许记录日志后继续抛出异常，因为这样会导致一次错误被多次记录

catch (IOException e) {

LOG.error("xxxx", e);

throw new Exception("xxx"); // 错误

}

7. 日志中尽量避免拼接字符串，如有函数调用操作要先做判断，因为即使当前级别高于debug，表达式也会被执行

LOG.debug("returning content: " + doSomething()); // 错误

// 正确

if (LOG.isDebugEnabled()) {

LOG.debug("returning content: {}", doSomething());

}