求你了，别乱打印日志了！

Java之道 3天前

以下文章来源于捡田螺的小男孩 ，作者捡田螺的小男孩

捡田螺的小男孩

专注后端技术栈，热爱分享，热爱交朋友，热爱工作总结。毕业于华南理工大学，软件...

前言

日志是快速定位问题的好帮手，是撕逼和甩锅的利器！打印好日志非常重要。今天我们来聊聊日志打印的15个好建议~

1. 选择恰当的日志级别

常见的日志级别有5种，分别是error、warn、info、debug、trace。日常开发中，我们需要选择恰当的日志级别，不要反手就是打印info哈~

- error：错误日志，指比较严重的错误，对正常业务有影响，需要运维配置监控的；
- warn：警告日志，一般的错误，对业务影响不大，但是需要开发关注；
- info：信息日志，记录排查问题的关键信息，如调用时间、出参入参等等；
- debug：用于开发DEBUG的，关键逻辑里面的运行时数据；
- trace：最详细的信息，一般这些信息只记录到日志文件中。

2. 日志要打印出方法的入参、出参

我们并不需要打印很多很多日志，只需要打印可以快速定位问题的有效日志。有效的日志，是甩锅的利器！

哪些算得的上有效关键的日志呢？比如说，方法进来的时候，打印入参。再然后呢，在方法返回的时候，就是打印出参，返回值。入参的话，一般就是**userId**或者**bizSeq**这些关键信息。正例如下：

```
public String testLogMethod(Document doc, Mode mode){
   log.debug(“method enter param: {}”,userId);
   String id = "666";
   log.debug(“method exit param: {}”,id);
   return id;
}
```

3. 选择合适的日志格式

理想的日志格式，应当包括这些最基本的信息：如当前时间戳（一般毫秒精确度）、日志级别，线程名字等等。在logback日志里可以这么配置：

```
<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
        <pattern>%d{HH:mm:ss.SSS} %-5level [%thread][%logger{0}] %m%n</pattern>
    </encoder>
</appender>
```

如果我们的日志格式，连当前时间都没有记录，那连请求的时间点都不知道了？

4. 遇到if...else...等条件时，每个分支首行都尽量打印日志

当你碰到**if...else...**或者**switch**这样的条件时，可以在分支的首行就打印日志，这样排查问题时，就可以通过日志，确定进入了哪个分支，代码逻辑更清晰，也更方便排查问题了。

正例：

```
if(user.isVip()){
  log.info("该用户是会员,Id:{},开始处理会员逻辑",user,getUserId());
  //会员逻辑
}else{
  log.info("该用户是非会员,Id:{},开始处理非会员逻辑",user,getUserId())
  //非会员逻辑
}
```

5.日志级别比较低时，进行日志开关判断

对于trace/debug这些比较低的日志级别，必须进行日志级别的开关判断。

正例：

```
User user = new User(666L, "公众号", "捡田螺的小男孩");
if (log.isDebugEnabled()) {
    log.debug("userId is: {}", user.getId());
}
```

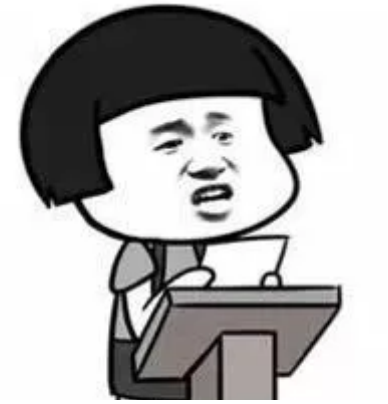
因为当前有如下的日志代码：

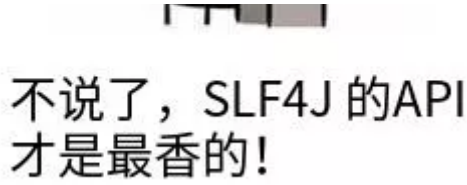
```
logger.debug("Processing trade with id: " + id + " and symbol: " + symbol);
```

如果配置的日志级别是**warn**的话，上述日志不会打印，但是会执行字符串拼接操作，如果 `symbol` 是对象， 还会执行 `toString()` 方法，浪费了系统资源，执行了上述操作，最终日志却没有打印，因此建议加日志开关判断。

6.不能直接使用日志系统（Log4j、Logback）中的 API，而是使用日志框架SLF4J中的API。

SLF4J 是门面模式的日志框架，有利于维护和各个类的日志处理方式统一，并且可以在保证不修改代码的情况下，很方便的实现底层日志框架的更换。





正例：

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

private static final Logger logger = LoggerFactory.getLogger(TianLuoBoy.class);
```

7. 建议使用参数占位{}，而不是用+拼接。

反例：

```
logger.info("Processing trade with id: " + id + " and symbol: " + symbol);
```

上面的例子中，使用 + 操作符进行字符串的拼接，有一定的性能损耗。

正例如下：

```
logger.info("Processing trade with id: {} and symbol : {} ", id, symbol);
```

我们使用了大括号 {} 来作为日志中的占位符，比于使用 + 操作符，更加优雅简洁。并且，相对于反例，使用占位符仅是替换动作，可以有效提升性能。

8. 建议使用异步的方式来输出日志。

- 日志最终会输出到文件或者其它输出流中的，IO性能会有要求的。如果异步，就可以显著提升IO性能。
- 除非有特殊要求，要不然建议使用异步的方式来输出日志。以logback为例吧，要配置异步很简单，使用AsyncAppender就行

```
<appender name="FILE_ASYNC" class="ch.qos.logback.classic.AsyncAppender">
```

```
    <appender-ref ref="ASYNC"/>
</appender>
```

9. 不要使用e.printStackTrace()

反例：

```
try{
  // 业务代码处理
}catch(Exception e){
  e.printStackTrace();
}
```

正例：

```
try{
  // 业务代码处理
}catch(Exception e){
  log.error("你的程序有异常啦",e);
}
```

理由：

- e.printStackTrace()打印出的堆栈日志跟业务代码日志是交错混合在一起的，通常排查异常日志不太方便。

- e.printStackTrace()语句产生的字符串记录的是堆栈信息，如果信息太长太多，字符串常量池所在的内存块没有空间了,即内存满了，那么，用户的请求就卡住啦~

10. 异常日志不要只打一半，要输出全部错误信息

反例1：

```
try {
    //业务代码处理
} catch (Exception e) {
    // 错误
    LOG.error('你的程序有异常啦');
}
```

- 异常e都没有打印出来，所以压根不知道出了什么类型的异常。

反例2：

```
try {
    //业务代码处理
} catch (Exception e) {
    // 错误
    LOG.error('你的程序有异常啦', e.getMessage());
}
```

- `e.getMessage()` 不会记录详细的堆栈异常信息，只会记录错误基本描述信息，不利于排查问题。

正例：

```
try {
    //业务代码处理
} catch (Exception e) {
    // 错误
    LOG.error('你的程序有异常啦', e);
}
```

11. 禁止在线上环境开启 debug

禁止在线上环境开启debug，这一点非常重要。

因为一般系统的debug日志会很多，并且各种框架中也大量使用 debug的日志，线上开启debug不久可能会打满磁盘，影响业务系统的正常运行。

12.不要记录了异常，又抛出异常

反例如下：

```
log.error("IO exception", e);
throw new MyException(e);
```

- 这样实现的话，通常会把栈信息打印两次。这是因为捕获了MyException异常的地方，还会再打印一次。
- 这样的日志记录，或者包装后再抛出去，不要同时使用！否则你的日志看起来会让人很迷惑。

13.避免重复打印日志

避免重复打印日志，酱紫会浪费磁盘空间。如果你已经有一行日志清楚表达了意思，避免再冗余打印，反例如下：

```
if(user.isVip()){
  log.info("该用户是会员,Id:{}",user,getUserId());
  //冗余，可以跟前面的日志合并一起
  log.info("开始处理会员逻辑,id:{}",user,getUserId());
  //会员逻辑
}else{
  //非会员逻辑
}
```

如果你是使用log4j日志框架，务必在 `log4j.xml` 中设置 additivity=false，因为可以避免重复打印日志

正例：

```
<logger name="com.taobao.dubbo.config" additivity="false">
```

14.日志文件分离

- 我们可以把不同类型的日志分离出去，比如access.log，或者error级别error.log，都可以单独打印到一个文件里面。
- 当然，也可以根据不同的业务模块，打印到不同的日志文件里，这样我们排查问题和做数据统计的时候，都会比较方便啦。

15. 核心功能模块，建议打印较完整的日志

- 我们日常开发中，如果核心或者逻辑复杂的代码，建议添加详细的注释，以及较详细的日志。
- 日志要多详细呢？脑洞一下，如果你的核心程序哪一步出错了，通过日志可以定位到，那就可以啦。

完

往期推荐

面试官：Redis的事务满足原子性吗？

在华为做外包的那些年...

当 Transactional 碰到锁，有个大坑！

有道无术，术可成；有术无道，止于术

欢迎大家关注Java之道公众号

好文章，我在看♡

喜欢此内容的人还喜欢

Lombok！代码简洁神器还是代码“亚健康”元凶？

ImportNew

为了偷懒，我们做了Yapi生成Typescript接口请求工具

高级前端进阶

Redis只能做缓存？太out了！

漫话编程