Java 程序如何正确的打日志

什么是日志

简单的说，日志就是记录程序的运行轨迹，方便查找关键信息，也方便快速定位解决问题。

我们 Java 程序员在开发项目时都是依赖 Eclipse/ Idea 等开发工具的 Debug 调试功能来跟踪解决 Bug，在开发环境可以这么做，但项目发布到了测试、生产环境呢？你有可能会说可以使用远程调试，但实际并不能允许让你这么做。

所以，日志的作用就是在测试、生产环境没有 Debug 调试工具时开发、测试人员定位问题的手段。日志打得好，就能根据日志的轨迹快速定位并解决线上问题，反之，日志输出不好不能定位到问题不说反而会影响系统的性能。

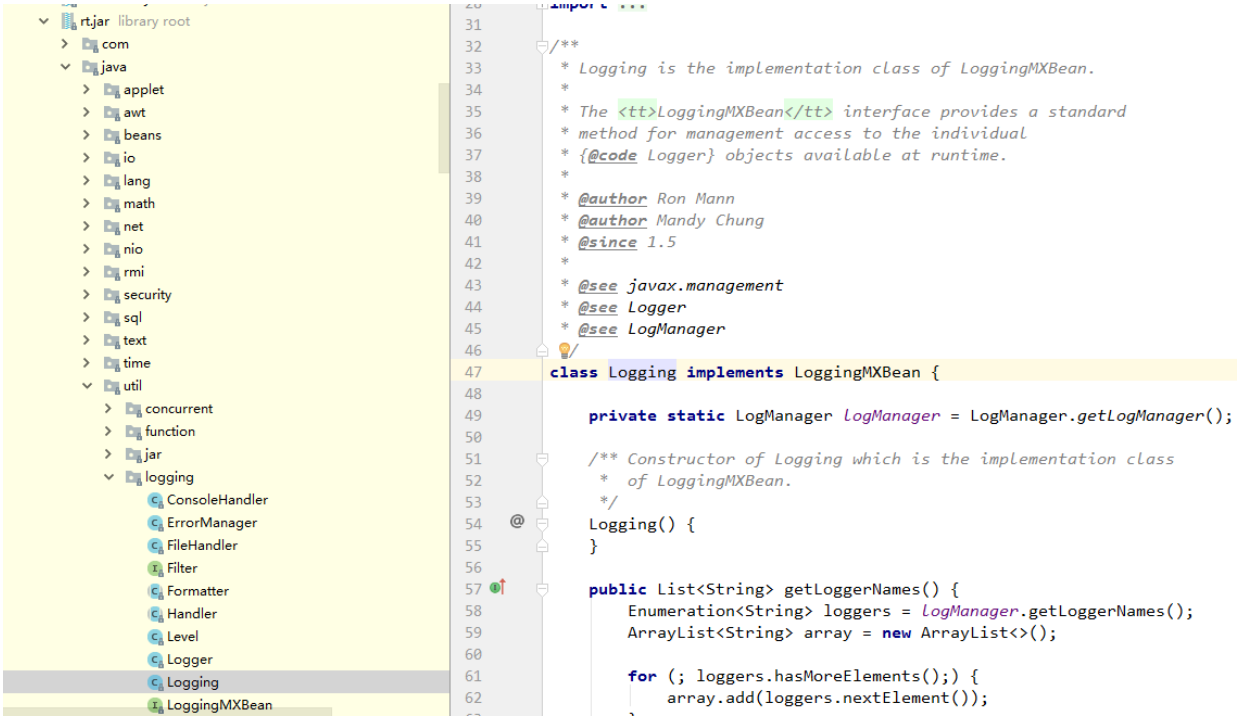
优秀的项目都是能根据日志定位问题的，而不是在线调试，或者半天找不到有用的日志而抓狂…

常用日志框架

log4j、Logging、commons-logging、slf4j、logback，开发的同学对这几个日志相关的技术不陌生吧，为什么有这么多日志技术，它们都是什么区别和联系呢？相信大多数人搞不清楚它们的关系，下面我将一一介绍一下，以后大家再也不用傻傻分不清楚了。

Logging

如图所示，这是 Java 自带的日志工具类，在 JDK 1.5 开始就已经有了，在 `java.util.logging` 包下。



> 更多关于 Java Logging 的介绍可以看官方文档

Log4j

Log4j 是 Apache 的一个开源日志框架，也是市场占有率最多的一个框架。大多数没用过 Java Logging， 但没人敢说没用过 Log4j 吧，反正从我接触 Java 开始就是这种情况，做 Java 项目必有 Log4j 日志框架。

注意：log4j 在 2015/08/05 这一天被 Apache 宣布停止维护了，用户需要切换到 Log4j2上面去。

下面是官方宣布原文：

> On August 5, 2015 the Logging Services Project Management Committee announced that Log4j 1.x had reached end of life. For complete text of the announcement please see the Apache Blog. Users of Log4j 1 are recommended to upgrade to Apache Log4j 2.
>
> Log4j2 的官方地址

commons-logging

上面介绍的 log4j 是一个具体的日志框架的实现，而 commons-logging 就是日志的门面接口，它也是 apache 最早提供的日志门面接口，用户可以根据喜好选择不同的日志实现框架，而不必改动日志定义，这就是日志门面的好处，符合面对接口抽象编程。

> 更多的详细说明可以参考官方说明

Slf4j

全称：Simple Logging Facade for Java，即简单日志门面接口，和 Apache 的 commons-logging 是一样的概念，它们都不是具体的日志框架，你可以指定其他主流的日志实现框架。

> Slf4j 的官方地址

Slf4j 也是现在主流的日志门面框架，使用 Slf4j 可以很灵活的使用占位符进行参数占位，简化代码，拥有更好的可读性，这个后面会讲到。

Logback

Logback 是 `Slf4j` 的原生实现框架，同样也是出自 `Log4j` 一个人之手，但拥有比 `log4j` 更多的优点、特性和更做强的性能，现在基本都用来代替 `log4j` 成为主流。

> Logback 的官方地址

为什么 Logback 会成为主流?

无论从设计上还是实现上，Logback相对log4j而言有了相对多的改进。不过尽管难以一一细数，这里还是列举部分理由为什么选择logback而不是log4j。牢记logback与log4j在概念上面是很相似的，它们都是有同一群开发者建立。所以如果你已经对log4j很熟悉，你也可以很快上手logback。如果你喜欢使用log4j，你也许会迷上使用logback。

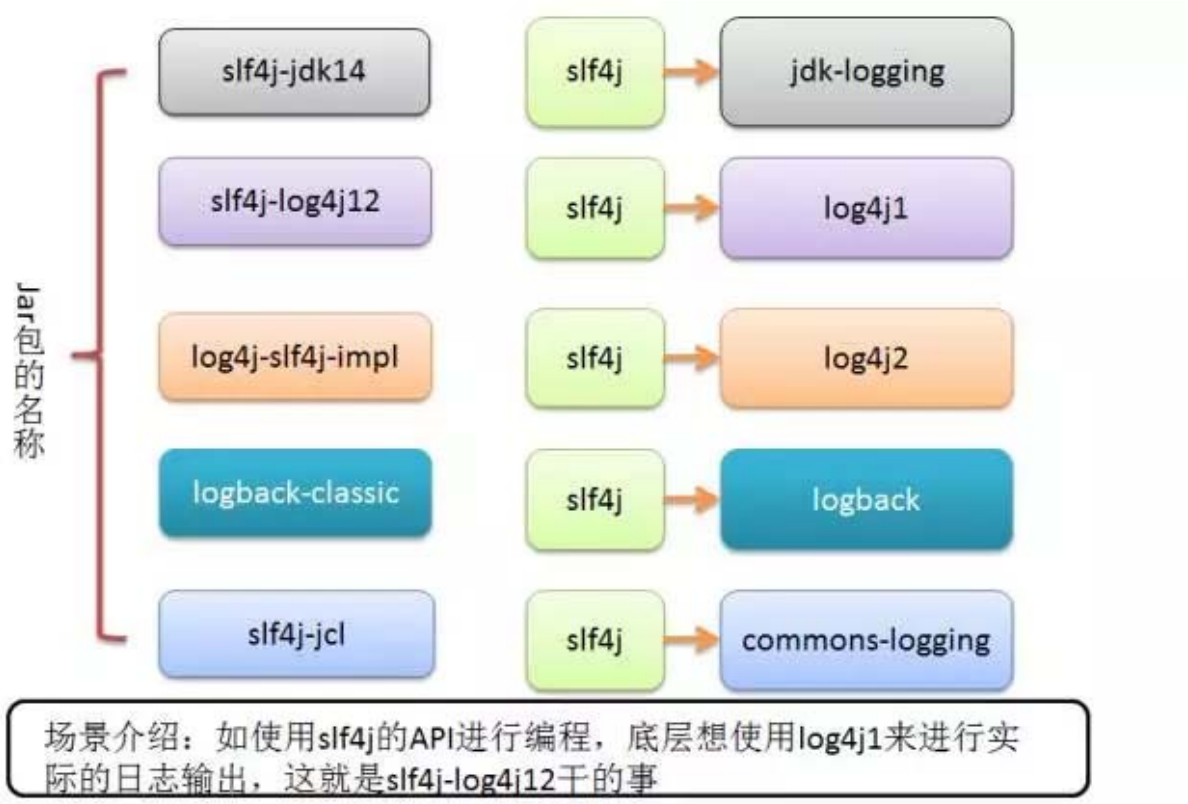
更快的执行速度

基于我们先前在log4j上的工作，logback 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。在保证logback的组件更加快速的同时，同时所需的内存更加少。

更多请参考《从Log4j迁移到LogBack的理由 》

日志框架总结

1. `commons-loggin` 、 `slf4j` 只是一种日志抽象门面，不是具体的日志框架。
2. `log4j` 、 `logback` 是具体的日志实现框架。
3. 一般首选强烈推荐使用 `slf4j + logback` 。当然也可以使用 `slf4j + log4j` 、`commons-logging + log4j` 这两种日志组合框架。

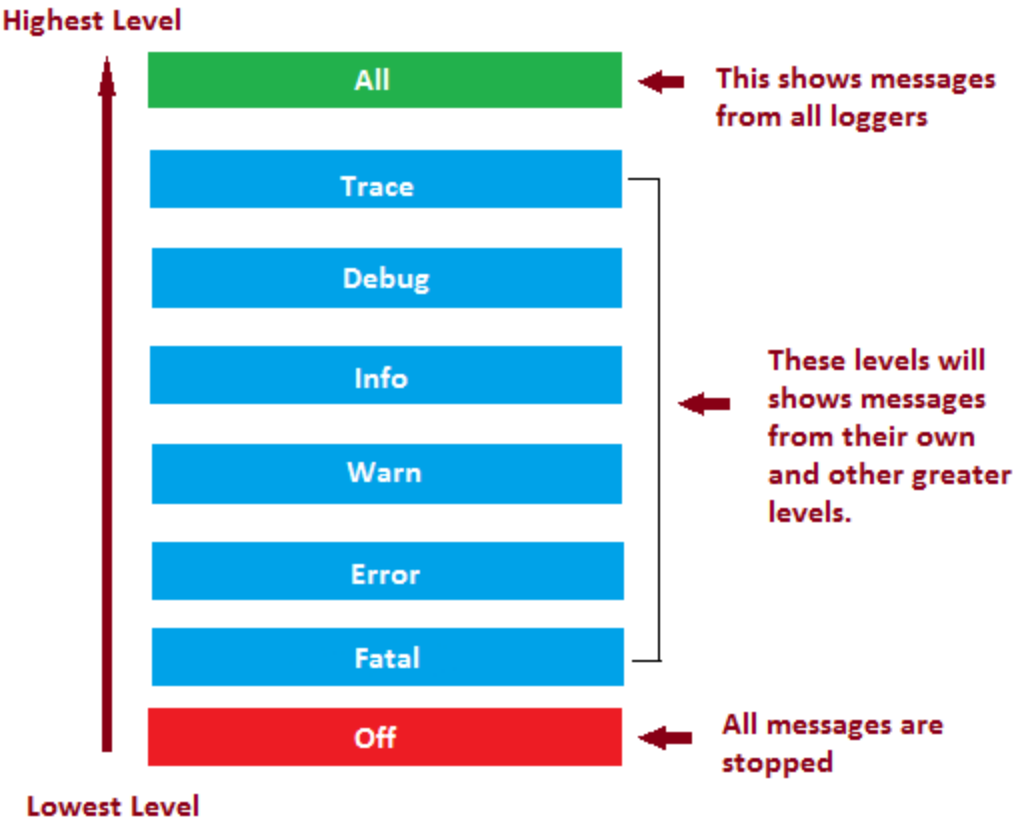


从上图可以看出 `slf4j` 很强大吧，不但能和各种日志框架对接，还能和日志门面 `commons-logging` 进行融合。

日志级别详解

日志的输出都是分级别的，不同的设置不同的场合打印不同的日志。下面拿最普遍用的 `Log4j` 日志框架来做个日志级别的说明，这个也比较奇全，其他的日志框架也都大同小异。

Log4j 的级别类 `org.apache.log4j.Level` 里面定义了日志级别，日志输出优先级由高到底分别为以下8种。



Log4j Levels - Preferences

日志级别	描述
OFF	关闭：最高级别，不输出日志。
FATAL	致命：输出非常严重的可能会导致应用程序终止的错误。
ERROR	错误：输出错误，但应用还能继续运行。
WARN	警告：输出可能潜在的危险状况。
INFO	信息：输出应用运行过程的详细信息。
DEBUG	调试：输出更细致的对调试应用有用的信息。
TRACE	跟踪：输出更细致的程序运行轨迹。
ALL	所有：输出所有级别信息。

所以，日志优先级别标准顺序为：

> ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL < OFF

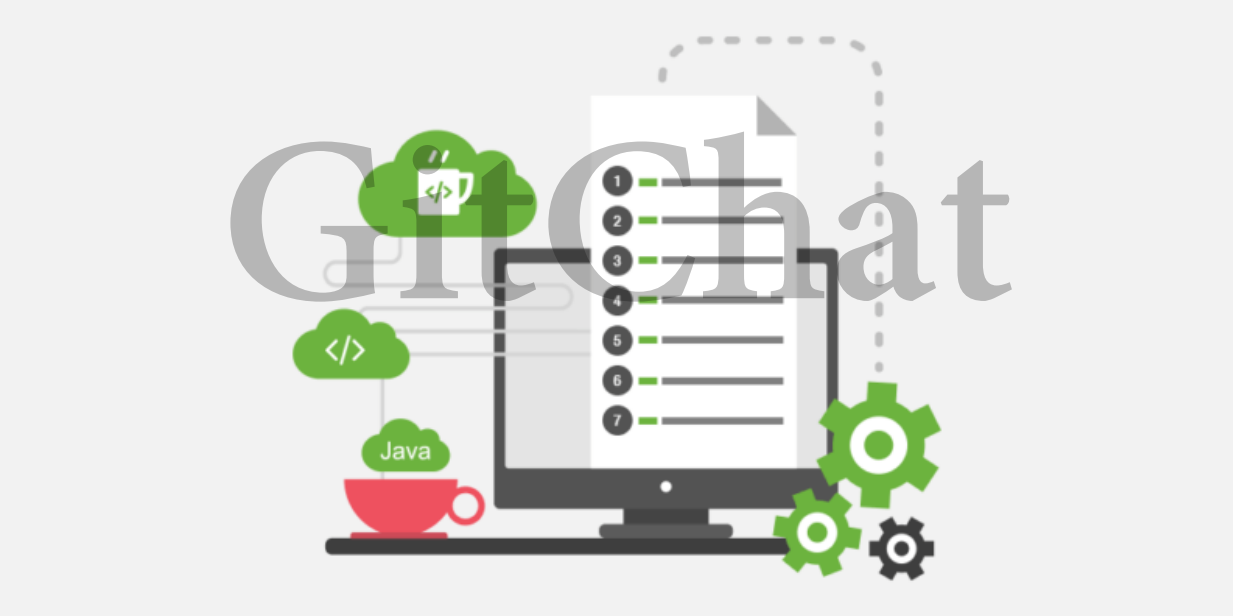
如果日志设置为 `L` ，一个级别为 `P` 的输出日志只有当 `P >= L` 时日志才会输出。

即如果日志级别 L 设置 INFO，只有 P 的输出级别为 INFO、WARN，后面的日志才会正常输出。

具体的输出关系可以参考下图：

			x: Visible				
	FATAL	ERROR	WARN	INFO	DEBUG	TRACE	ALL
OFF							
FATAL	x						
ERROR	x	x					
WARN	x	x	x				
INFO	x	x	x	x			
DEBUG	x	x	x	x	x		
TRACE	x	x	x	x	x	x	
ALL	x	x	x	x	x	x	x

知道了日志级别，这还只是基础，如何了解打日志的规范，以及如何正确地打日志姿势呢？！



打日志的规范准则

最开始也说过了，日志不能乱打，不然起不到日志本应该起到的作用不说，还会造成系统的负担。在 BAT、华为一些大公司都是对日志规范有要求的，什么时候该打什么日志都是有规范的。

阿里去年发布的《Java 开发手册》，里面有一章节就是关于日志规范的，让我们再来回顾下都有什么内容。

下面是阿里的《Java开发手册》终极版日志规约篇。

规范有很多，这里就不再一一详述了，完整终极版可以在微信公众号 **“Java 技术栈”** 中回复 **“手册”** 获取。这里只想告诉大家，在大公司打日志都是有严格规范的，不是你随便打就行的。

阿里是一线互联网公司，所制定的日志规范也都符合我们的要求，很有参考意义，能把阿里这套日志规约普及也真很不错了。

项目中该如何正确的打日志？

1. 正确的定义日志


```
private static final Logger LOG =
LoggerFactory.getLogger(this.getClass());
```

通常一个类只有一个 LOG 对象，如果有父类可以将 LOG 定义在父类中。

日志变量类型定义为门面接口（如 slf4j 的 Logger），实现类可以是 `Log4j`、`Logback` 等日志实现框架，不要把实现类定义为变量类型，否则日志切换不方便，也不符合抽象编程思想。

2、使用参数化形式 `{}` 占位，`[]` 进行参数隔离

```
LOG.debug("Save order with order no: [{}], and order amount:
[{}]");
```

这种可读性好，这样一看就知道 `[]` 里面是输出的动态参数，`{}` 用来占位类似绑定变量，而且只有真正准备打印的时候才会处理参数，方便定位问题。

如果日志框架不支持参数化形式，且日志输出时不支持该日志级别时会导致对象冗余创建，浪费内存，此时就需要使用 `isXXEnabled` 判断，如：

```
if(LOG.isDebugEnabled()){
    // 如果日志不支持参数化形式，debug又没开启，那字符串拼接就是无用的代码
拼接，影响系统性能
```

```
        logger.debug("Save order with order no: " + orderNo + ", and
    order amount: " + orderAmount);
    }
```

至少 debug 级别是需要开启判断的，线上日志级别至少应该是 info 以上的。

这里推荐大家用 SLF4J 的门面接口，可以用参数化形式输出日志， debug 级别也不必用 if 判断，简化代码。

3、输出不同级别的日志

项目中最常用有日志级别是 ERROR 、 WARN 、 INFO 、 DEBUG 四种了，这四个都有怎样的应用场景呢。

- **ERROR（错误）**

一般用来记录程序中发生的任何异常错误信息（Throwable），或者是记录业务逻辑出错。

- **WARN（警告）**

一般用来记录一些用户输入参数错误、

- **INFO（信息）**

这个也是平时用的最低的，也是默认的日志级别，用来记录程序运行中的一些有用的信息。如程序运行开始、结束、耗时、重要参数等信息，需要注意有选择性的有意义的输出，到时候自己找问题看一堆日志却找不到关键日志就没意义了。

- **DEBUG（调试）**

这个级别一般记录一些运行中的中间参数信息，只允许在开发环境开启，选择性在测试环境开启。

几个错误的打日志方式

1. 不要使用 System.out.print..

输出日志的时候只能通过日志框架来输出日志，而不能使用 System.out.print.. 来打印日志，这个只会打印到 tomcat 控制台，而不会记录到日志文件中，不方便管理日志，如果通过服务形式启动把日志丢弃了那更是找不到日志了。

2. 不要使用 e.printStackTrace()

首先来看看它的源码：

```
public void printStackTrace() {
    printStackTrace(System.err);
}
```

它其实也是利用 System.err 输出到了 tomcat 控制台。

3. 不要抛出异常后又输出日志

如捕获异常后又抛出了自定义业务异常，此时无需记录错误日志，由最终捕获方进行异常处理。不能又抛出异常，又打印错误日志，不然会造成重复输出日志。

```
try {
    // ...
} catch (Exception e) {
    // 错误
    LOG.error("xxx", e);
    throw new RuntimeException();
}
```

4. 不要使用具体的日志实现类


```
InterfaceImpl interface = new InterfaceImpl();
```

这段代码大家都看得懂吧？应该面向接口的对象编程，而不是面向实现，这也是软件设计模式的原则，正确的做法应该是。

```
Interface interface = new InterfaceImpl();
```

日志框架里面也是如此，上面也说了，日志有门面接口，有具体实现的实现框架，所以大家不要面向实现编程。

5. 没有输出全部错误信息

看以下代码，这样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。

```
try {
    // ...
} catch (Exception e) {
    // 错误
```

```
        LOG.error('XX 发生异常', e.getMessage());

        // 正确
        LOG.error('XX 发生异常', e);
    }
```

6. 不要使用错误的日志级别

曾经在线上定位一个问题，同事自信地和我说：明明输出了日志啊，为什么找不到...，后来我去看了下他的代码，是这样的：

```
    try {
        // ...
    } catch (Exception e) {
        // 错误
        LOG.info("XX 发生异常...", e);
    }
```

大家看出了什么问题吗？用 info 记录 error 日志，日志输出到了 info 日志文件中了，同事拼命地在 error 错误日志文件里面找怎么能找到呢？

7. 不要在千层循环中打印日志

这个是什么意思，如果你的框架使用了性能不高的 Log4j 框架，那就不要在上千个 for 循环中打印日志，这样可能会拖垮你的应用程序，如果你的程序响应时间变慢，那要考虑是不是日志打印的过多了。

```
    for(int i=0; i<2000; i++){
        LOG.info("XX");
    }
```

最好的办法是在循环中记录要点，在循环外面总结打印出来。

8. 禁止在线上环境开启 debug

这是最后一点，也是最重要的一点。

一是因为项目本身 debug 日志太多，二是各种框架中也大量使用 debug 的日志，线上开启 debug 不久就会打满磁盘，影响业务系统的正常运行。

好了，关于 Java 如何正确的打日志差不多就到这里了，我的见识和经历有限，不一定全部正确，或者有遗漏的地方。如果你有疑问或者更好的建议欢迎在下面给我留言，一起学习探讨。

最后，作者的官方博客（blog.javastack.cn），微信公众号（Java技术栈），大家后续可以在这两个平台上面和我交流