

Java 日志从入门到实战

之前分享了一篇《Java 异常处理从入门到实战》，当时有人提出如果能分享一篇日志处理相关的文章就更好了。这篇 Chat 就是对之前参加异常处理 Chat 同学的一个回应，希望能带给大家一些收获。

日志和异常处理结合得当的话，会给项目维护带来非常大的价值。

- **日志**：就是介绍一个过程和经历的详细记录。
- **项目日志**：就是项目开发过程的详细记录，一般由项目经理记录。
- **代码里的日志**：就是程序员记录某个开发过程的详细情况，这是项目里每个程序员需要做的工作。

代码里的日志在项目中扮演着非常重要的角色，日志记录的详细程度决定系统是否容易维护，Java 项目中日志记录的框架有很多，在项目中如何选型也必将困惑。

我们这场 chat 讨论的主题就是：**Java 代码里的日志**，主要是写给 Java 程序员看的，本篇 Chat 从以下六个方面对 Java 日志进行阐述：

- 一、日志在项目中的作用
- 二、Java 日志使用的困惑
- 三、Java 日志演化历史
- 四、使用不同框架的 Java 日志示例说明
- 五、项目中日志记录遵守哪些原则
- 六、一个推荐的项目 Java 日志实例

以期通过本场 Chat 的学习，能使大家在编码过程中，对日志的处理过程加深认识、统一规范、得到收获，从零开始轻松掌握 Java 日志处理，并且可以运用到实际项目中。

一、日志在项目中的作用

Log 日志，主要用于记录程序运行的情况，以便于程序在部署之后的排错调试等，也有利于将这些信息进行持久化（如果不将日志信息保存到文件或数据库，则信息便会丢失）。

1. 查看程序当前运行状态

如果想了解程序当前的运行情况，我们通过实时查看应用日志的输出，就能进行分析。

比如，你在浏览器里输入一个 action 地址，该 url 负责执行一些批量处理，action 运行后，假设处理比较耗时，你再浏览器里无法直接看到程序的执行结果，此时，你可以打开系统日志，通过从日志输出信息就能轻松地分析该 url 的执行情况。

2. 查看程序历史运行轨迹

如果想了解历时程序的运行情况，我们通过查看应用历时日志的输出，就能进行分析。

比如，你想了解下上周周末用户访问量，你可以打开系统上周周末的日志记录，进行分析。你想了解昨天的某个定时任务是否正常执行，你可以打开昨天的系统日志，精确查找该定时任务的输出信息，从而判断定时任务是否执行。

3. 排查系统问题

排查系统问题是程序员最熟悉的味道了，在项目维护过程中，出了任何问题，都需要程序员去进行排查。此时，如果没有清楚明了的日志记录，想要核查出问题的原因，难于上青天。

一个优秀的程序员一定是个日志记录高手，如果日志记录的好，处理得当，排查问题则易如反掌。

大家有没有遇到一种场景，一个问题发生了，有的人能迅速定位问题并解决，有的人搞了半天，还没发现问题的产生原因。

其实快速定位问题的人一定记录了详细的日志，因此当问题发生的时候，通过核查问题发生时候的日志，就能快速的找出问题产生的原因。

4. 优化系统性能

通过记录程序运行的时间，就能判断程序从执行开始到执行结束消耗的时间，从而判断系统性能是否达标，为系统性能优化提供判断依据。

5. 安全审计的基石

网络安全越来越受到大家的关注，所以系统安全目前是项目过程非常重要的一个环节，安全审计也是系统中非常重要的部分。

通过系统日志分析，可以判断一些非法攻击，非法调用，以及系统处理过程中的安全隐患。

比如，大家平时都在做运营系统，其中运营人员在通过界面处理一些数据的时候，如果没有清楚的日志操作记录，一条数据被删除或者修改，你是无法找到是谁操作的，但是如果你做了相应的记录，该数据被谁删除或者修改就会一目了然。

通过以上 5 点说明了日志在项目维护过程中的重要作用

一个系统是否容易维护，很大程度上是基于程序员在程序开发过程中的代码日志是怎么记录的。

日志记录的越清楚，维护起来就越容易，有的程序员没有日志记录意识，或者对日志记录认识不清，或者是不知道日志该如何记录，这势必会给项目后期的维护带来一个个大坑。

当项目经理让你解决一个线上问题的时候，正好遇到了一个没有日志记录习惯的人写的代码，你就能体会到那种痛苦，不由地想要爆粗口。

因此，作为一个程序员来说，掌握代码日志的记录方法，是程序员生涯的一项基本功。写代码时做好日志记录是“即利人又利己”的做法，不写日志记录就是“损人不利己”的做法。

二、Java 日志使用的困惑

大多数的程序员都能认识到日志在项目中的重要性，可是对日志记录具体要怎么做，做到什么程度，日志记录用什么工具，会有很多困惑。

1. 工具困惑

作为 Java 程序员，幸运的是，Java 拥有功能和性能都非常强大的日志库；不幸的是，这样的日志库有不只一个，相信每个人都会对 JUL(Java Util Log)、JCL(Commons Logging)、Log4j、SLF4J、Logback、Log4j2 等等的日志工具，到底使用什么产生感到困扰。

下面的第三节：“Java 日志演化历史”，第四节：“使用不同框架的 Java 日志示例说明”，为大家解惑 Java 日志框架的使用问题。

2. 使用困惑

有的程序员即使知道写 Java 程序用什么日志工具，可能对日志记录具体应该怎么写，写什么东西，什么情况下要写，这些仁者见仁智者见智的东西也会产生困扰。

下面的第五节：“项目中日志记录遵守哪些原则”，为大家解惑日志记录要遵守哪些原则。

有的程序员知道了该用哪个 Java 框架，也知道了日志记录的原则，可能还会对具体在项目中到底该如何操作产生困惑。

3. 实战困惑

下面的第六节：“一个推荐的项目 Java 日志实例”，为大家展示在具体项目过程中的应用示例。

三、Java 日志演化历史

最先出现的是 Apache 开源社区的 log4j，这个日志确实是应用最广泛的日志工具，成为了 Java 日志的事实上的标准。

然而，当时 Java 的开发主体 Sun 公司认为自己才是正统，在 Jdk1.4 中增加了 JUL（在 `java.util.logging` 包下）日志实现，企图对抗 log4j，但是却造成了 Java 目前开发者记录日志局面的混乱，迄今为止仍饱受诟病。

当然也有其他日志工具的出现，基本都是各自为政，这些日志系统互相没有关联。

为什么 JUL 的出现会导致开发局面混乱呢？

想象下你的项目应用使用 log4j，然后使用了一个第三方库，而第三方库使用了 JUL，那么，你的应用就得同时使用 log4j 和 JUL 两个日志工具了，然后又有需要使用另外一个第三方库，但是这个第三方库使用了 log4j 和 JUL 之外的 simplelog。这个时候你的应用里各种 log 工具满天飞，这势必会使你的程序员感到崩溃。因为这些日志工具互相没有关联，替换和统一日志工具也就变成了比较棘手的一件事情。

如果你遇到了这种问题，你该如何解决呢？

解决这个问题，我们会用到一个设计模式——“适配器模式”，即把这个问题进行抽象，抽象出一个接口层，对每个日志实现都进行适配，这样这些提供给别人的库都直接使用抽象的接口层即可。

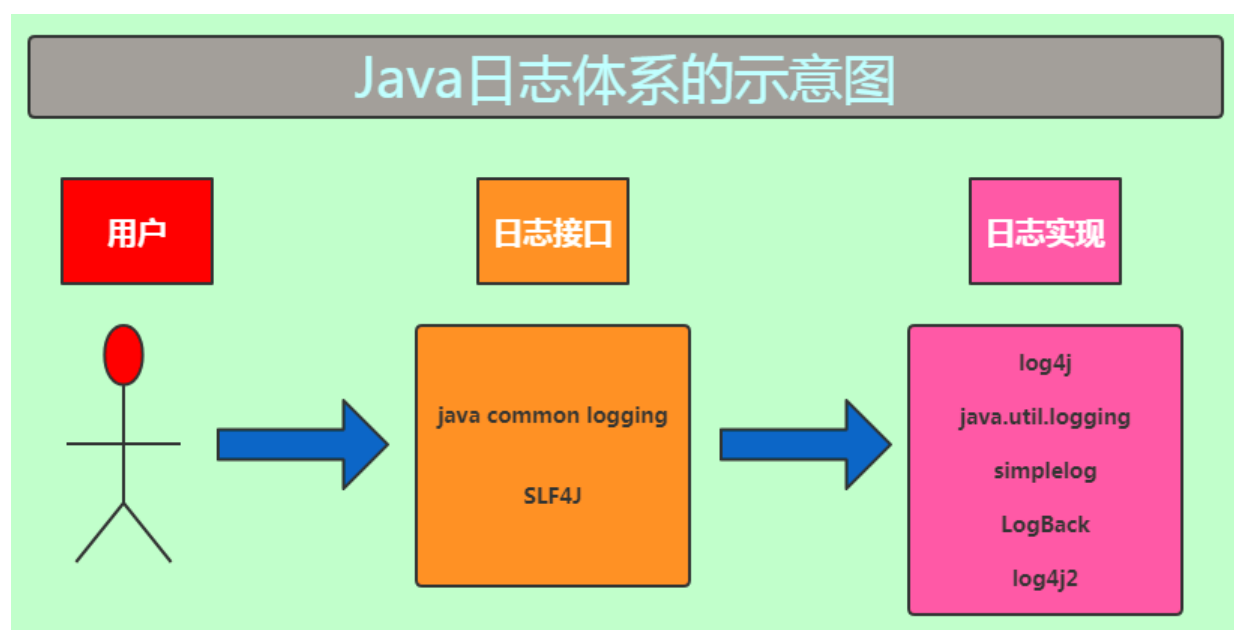
为了搞定这个日常开发中比较棘手的问题，Apache 开源社区提供了一个日志框架作为日志的抽象，叫 `commons-logging`，也被称为 JCL(`java common logging`)，JCL 对各种日志接口进行抽象，抽象出一个接口层，对每个日志实现都进行适配，这样这些提供给别人的库都直接使用抽象层即可，确实出色地完成了兼容主流的日志实现（log4j、JUL、simplelog 等），较好的解决了上述问题，基本一统江湖，就连顶顶大名的 spring 也是依赖了 JCL。

但是美好的日子并不长，作为元老级日志 log4j 的作者 (Ceki Gülcü)，他觉得 JCL 不够优秀，所以他再度出山，搞出了一套更优雅的日志框架 SLF4J（这个也是抽象层），即简单日志门面（`Simple Logging Facade for Java`），并为 SLF4J 实现了一个亲儿子——logback，确实更加优雅了。

最后，Ceki Gülcü 觉得还是得照顾下自己的“大儿子”——log4j，又把 log4j 进行了改造，就是所谓的 log4j2，同时支持 JCL 以及 SLF4J。

SLF4J 的出现，又使 Java 日志体系变得混乱起来。

下面是一张目前 Java 日志体系的示意图：



日志库 log4j，JUL，logback 是互相不兼容的，没有共同的 Interface，所以 commons-logging、SLF4J 通过适配器模式，抽象出来一个共同的接口，然后根据使用的具体日志框架来实现日志。

java common logging 和 SLF4J 都是日志的接口，供用户使用，而没有提供实现，log4j，JUL，logback 等等才是日志的真正实现。

当我们调用日志接口时，接口会自动寻找恰当的实现，返回一个合适的实例给我们服务。这些过程都是透明化的，用户不需要进行任何操作。

工具	官方网站
----	------

log4j	http://logging.apache.org/log4j/1.2
-------	---

JCL	http://commons.apache.org/proper/commons-logging/
-----	---

SLF4J	http://www.slf4j.org
-------	---

logback	http://logback.qos.ch
---------	---

log4j2	https://logging.apache.org/log4j/2.x/
--------	---

四、使用不同框架的 Java 日志示例说明

1. log4j

log4j 介绍

Log4j (log for java) 是 Apache 的一个开源项目，通过使用 Log4j，我们可以控制日志信息输出到日志文件、也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，我们能够更加细致地控制日志的生成过程。这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用程序的代码。

log4j 使用步骤说明

(1) 选择 jar 包

加入 log4j-1.2.17.jar 到 Libraries 下，如果使用 maven 项目，也可以选择 pom.xml 中新增依赖如下：

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
```

(2) log4j.properties 配置

Log4j 有三个主要组件：记录器，appender 和布局。这三种类型的组件协同工作，使开发人员能够根据消息类型和级别记录消息，并在运行时控制这些消息的格式和报告位置。

Log4j 建议只使用四个级别，优先级从高到低分别是 ERROR、WARN、INFO、DEBUG。

比如定义了 INFO 级别，只有等于及高于这个级别的才进行处理，则应用程序中所有 DEBUG 级别的日志信息将不被打印出来。ALL，打印所有的日志；OFF，关闭所有的日志输出。

appenderName，就是指定日志信息输出到哪个地方。可同时指定多个输出目的地。

在 src 根目录下建立 log4j.properties，根据自己的需求，相应的修改其中的配置，其内容如下所示：

```
#配置根Logger
#改代码表示输输出info级别以上的日志，文件分别输出，一个是file，一个是
error
log4j.rootLogger=info,file,error

#配置file日志信息输出目的地Appender
#定义名为file的输出端是每天产生一个日志文件
log4j.appender.file=org.apache.log4j.DailyRollingFileAppender
#指定日志信息的最低输出级别位INFO，默认为DEBUG。
log4j.appender.file.Threshold=INFO
#指定当前消息输出到jpm/log4j/log.log文件中
log4j.appender.file.File=jpm/log4j/log.log
```

```

#指定按天来滚动日志文件
log4j.appender.file.DatePattern=yyyy-MM-dd
#配置日志信息的格式（布局）Layout是可以灵活地指定布局模式
log4j.appender.file.layout=org.apache.log4j.PatternLayout
#格式化日志，Log4J采用类似C语言中的printf函数的打印格式格式化日志信息
log4j.appender.file.layout.ConversionPattern=[%d{yyyy-MM-ddHH:mm:ss}][%-5p][jpm-%c{1}-%M(%L)]-%m%n
#指定输出信息的编码
log4j.appender.file.encoding=UTF-8

#配置error日志信息输出目的地Appender
#定义名为error的输出端是每天产生一个日志文件

log4j.appender.error=org.apache.log4j.DailyRollingFileAppender
#指定日志信息的最低输出级别位ERROR，默认为DEBUG。
log4j.appender.error.Threshold=ERROR
#指定当前消息输出到jpm/log4j/error.log文件中
log4j.appender.error.File=/jpm/log4j/error.log
#指定按月来滚动日志文件
log4j.appender.error.DatePattern=yyyy-MM
#配置日志信息的格式（布局）Layout是可以灵活地指定布局模式
log4j.appender.error.layout=org.apache.log4j.PatternLayout
#格式化日志，Log4J采用类似C语言中的printf函数的打印格式格式化日志信息
log4j.appender.error.layout.ConversionPattern=[%d{yyyy-MM-ddHH:mm:ss}][%-5p][jpm-%c{1}-%M(%L)]-%m%n
#指定输出信息的编码
log4j.appender.error.encoding=UTF-8

#使某个功能的日志单独输出到指定的日志文件
log4j.logger.saveUserLog=INFO,saveUserLog
#该配置就是让job的日志只输出到自己指定的日志文件中,表示Logger不会在父
Logger的appender里输出，默认为true。
log4j.additivity.saveUserLog=false

log4j.appender.saveUserLog=org.apache.log4j.DailyRollingFileAppender
log4j.appender.saveUserLog.File=/jpm/log4j/saveUserLog.log
log4j.appender.saveUserLog.DatePattern=yyyy-MM-dd
log4j.appender.saveUserLog.Append=true

log4j.appender.saveUserLog.layout=org.apache.log4j.PatternLayout
log4j.appender.saveUserLog.layout.ConversionPattern=%m%n
log4j.appender.error.encoding=UTF-8

```

(3) 输出日志的代码示例

```

package jpm;

import org.apache.log4j.Logger;

```

```

public class TestLog4j {

    public static void main(String[] args) {
        final Logger logger =
Logger.getLogger(TestLog4j.class);
        final Logger saveUserLog =
Logger.getLogger("saveUserLog");

        if (logger.isDebugEnabled()) {
            logger.debug("debug");
        }

        logger.info("info");
        logger.error("error");

        saveUserLog.info("张三,男,26岁,北京大学,2018-05-19,学
霸");

    }

}

```

(4) 以上代码示例生成的日志文件及日志详情

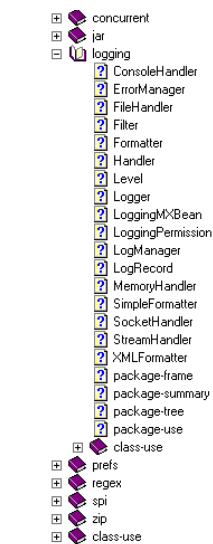


Log Entry	Log File
[2018-05-19 17:47:10] [INFO] [jpm-TestLog4j-main(15)]-info	log.log
[2018-05-19 17:47:10] [ERROR] [jpm-TestLog4j-main(16)]-error	error.log
[2018-05-19 17:47:10] [ERROR] [jpm-TestLog4j-main(16)]-error	error.log
张三,男,26岁,北京大学,2018-05-19,学霸	saveUserLog.log

2. JUL (在 java.util.logging 包下) 使用示例

JUL 介绍

Java Logging API 是 Sun 公司于 2002 年 5 月正式发布的。它是自 J2SE 1.4 版本开始提供的一个新的应用程序接口，需 JDK1.4 版本以上才能支持，java.util.logging.* 包是 JDK 的日志记录 API。



软件包 java.util.logging

提供 Java™ 2 平台核心日志工具的和接口。

请参见：
[描述](#)

接口摘要

Filter	Filter 可用于为记录内容提供比记录级别所提供的更细粒度的控制。
LoggingMXBean	日志记录设施的管理接口。

类摘要

ConsoleHandler	此 Handler 向 System.err 发布日志记录。
ErrorHandler	可将 ErrorHandler 对象附加到 Handler，以便处理日志记录过程中 Handler 上所发生的错误。
FileHandler	简单的文件日志记录 Handler。
Formatter	Formatter 为格式化 LogRecords 提供支持。
Handler	Handler 对象从 Logger 中获取日志信息，并将这些信息导出。
Level	Level 类定义了一组用来控制日志输出的标准日志级别。
Logger	Logger 对象用来记录特定系统或应用程序组件的日志消息。

强烈不推荐使用 java.util.logging 记录日志，因此这里不提供对应示例给大家。

3. java common logging 介绍

commons-logging 提供的是一个日志接口，是为那些需要建立在不同环境下使用不同日志架构的组件或库的开发者创建的，其中包括 log4j 以及 Java log 的日志架构。commons-logging 有两个基本的抽象类：Log(基本记录器) 和 LogFactory(负责创建 Log 实例)。把日志信息抽象成 commons-logging 的 Log 接口，并由 commons-logging 在运行时决定使用哪种日志架构。因为 Log4j 的强大功能，commons-logging 一般会与 Log4j 一起使用，这几乎成为了 Java 日志的标准工具。

4. SLF4J介绍

SLF4J 全称为 Simple Logging Facade for JAVA，即 java 简单日志门面。和 commons-logging 一样也是对不同日志框架提供的一个门面封装，可以在部署的时候不修改任何配置即可接入一种日志实现方案，能支持多个参数，并通过 {} 占位符进行替换。

看这个 Log4J 示例：

```
Logger.debug("Hello " + name);
```

由于字符串拼接的问题（注：上述语句会先拼接字符串，再根据当前级别是否低于 debug 决定是否输出本条日志，即使不输出日志，字符串拼接操作也会执行），因此许多公司一般强制使用下面的语句，这样只有当前处于DEBUG级别时才会执行字符串拼接：

```
if (logger.isDebugEnabled()) {  
    LOGGER.debug("Hello " + name);  
}
```

它避免了字符串拼接问题，可是有点太繁琐了。而 SLF4J 提供下面这样简单的语法：

```
LOGGER.debug("Hello {}", name);
```

它的形式类似第一条示例，但是又没有字符串拼接问题，也不像第二条那样繁琐。

正是因为 SLF4J 的这个占位符功能，使得人们越来越多的使用 SLF4J 这个接口用到实际开发项目中。

为什么需要日志接口，直接使用具体的实现不就行了吗？

接口用于定制规范，可以有多个实现，使用时是面向接口的（导入的包都是 SLF4J 的包或者是 JCL 的包，而不是具体某个日志框架中的包），即直接和接口交互，不直接使用实现，所以当需要更换实现的时候，直接更换实就可以了，而不用更改代码中的日志相关代码。

比如：SLF4J 定义了一套日志接口，项目中使用的日志框架是 log4j，开发中调用的所有接口都是 SLF4J 的，不直接使用 log4j，项目应用调用 SLF4J 的接口，SLF4J 的接口去调用 log4j 的实现，整个应用程序并没有直接使用 log4j，当项目需要更换更加优秀的日志框架时（如 logback）只需要引入 logback 的 jar 和 logback 对应的配置文件即可，完全不用更改 Java 代码中的日志相关的代码 logger.info("hello world")，也不用修改日志相关的类的导入的包（import org.slf4j.Logger; import org.slf4j.LoggerFactory;）。

因此日志门面的使用，为后续具体日志系统的实现更换带来了方便。

5. logback 使用示例

LogBack 介绍

LogBack 和 Log4j 都是开源日记工具库，LogBack 是 Log4j 的改良版本，比 Log4j 拥有更多的特性，同时也带来很大性能提升。LogBack 官方建议配合 Slf4j 使用，这样可以灵活地替换底层日志框架。

Logback 主要由三个模块组成：

- logback-core
- logback-classic
- logback-access

其中 logback-core 提供了 LogBack 的核心功能，是另外两个组件的基础。logback-classic 的地位和作用等同于 Log4J，它也被认为是 Log4J 的一个改进版，并且它实现了简单日志门面 SLF4J，所以当想配合 SLF4J 使用时，需要将 logback-classic 加入 classpath；而 logback-access 主要作为一个与 Servlet 容器交互的模块，比如说 tomcat 或者 jetty，提供一些与 HTTP 访问相关的功能。

logback 使用步骤说明

(1) 选择 jar 包

想在 Java 程序中使用 Logback，需要依赖三个 jar 包，分别是 slf4j-api，logback-core，logback-classic。其中 slf4j-api 并不是 Logback 的一部分，建议将 SLF4J 与 Logback 结合使用。

pom.xml

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.25</version>
</dependency>

<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-core</artifactId>
  <version>1.2.3</version>
</dependency>

<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.3</version>
</dependency>

<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-access</artifactId>
  <version>1.2.3</version>
</dependency>
```

(2) logback.xml

在 src 根目录下建立 logback.xml，根据自己的需求，相应的修改其中的配置，其内容如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 日志级别以及优先级排序：OFF > FATAL > ERROR > WARN > INFO >
DEBUG > TRACE > ALL -->
<!-- status用来指定log4j本身的打印日志的级别 -->
<!--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="WARN" monitorInterval="30">
  <!--先定义所有的appender -->
  <appenders>
    <!--这个输出控制台的配置 -->
    <console name="Console" target="SYSTEM_OUT">
      <!--输出日志的格式 -->
```

```

        <PatternLayout
            pattern="%d{yyyy-MM-dd HH:mm:ss.SSS}
[%-5level] %l %logger{36} - %msg%n" />
        </console>

<!--定义输出到指定位置的文件 -->
<File name="log" fileName="/jpm/log4j2/logs/log.log"
append="true">
    <PatternLayout
        pattern="%d{yyyy-MM-dd HH:mm:ss.SSS}
[%-5level] %l %logger{36} - %msg%n" />
    </File>

<!-- 这个会打印出所有的info及以下级别的信息，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 -->
    <RollingFile name="RollingFileInfo"
fileName="/jpm/log4j2/logs/info.log"
        filePattern="/jpm/log4j2/logs/${date:yyyy-
MM}/info-%d{yyyy-MM-dd}-%i.log">
        <!--控制台只输出level及以上级别的信息（onMatch），其他
的直接拒绝（onMismatch） -->
        <!-- DENY，日志将立即被抛弃不再经过其他过滤器；
NEUTRAL，有序列表里的下个过滤器过接着处理日志； ACCEPT，日志会被立即处理，
不再经过剩余过滤器。 -->
        <ThresholdFilter level="error" onMatch="DENY"
            onMismatch="ACCEPT" />
        <PatternLayout
            pattern="%d{yyyy-MM-dd HH:mm:ss.SSS}
[%-5level] %l %logger{36} - %msg%n" />
        <Policies>
            <TimeBasedTriggeringPolicy />
            <SizeBasedTriggeringPolicy size="100 MB" />
        </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30 -->
        <DefaultRolloverStrategy max="30" />
    </RollingFile>

    <RollingFile name="RollingFileError"
fileName="/jpm/log4j2/logs/error.log"
        filePattern="/jpm/log4j2/logs/${date:yyyy-
MM}/error-%d{yyyy-MM-dd}-%i.log">
        <ThresholdFilter level="ERROR" onMatch="ACCEPT"
            onMismatch="DENY" />
        <PatternLayout
            pattern="%d{yyyy-MM-dd HH:mm:ss.SSS}
[%-5level] %l %logger{36} - %msg%n" />
        <Policies>
            <TimeBasedTriggeringPolicy />
            <SizeBasedTriggeringPolicy size="100 MB" />
        </Policies>

```

```

        </RollingFile>
    </appenders>

    <!--只有定义了logger并引入的appender， appender才会生效 -->
    <loggers>
        <!--过滤掉spring和mybatis的一些无用的DEBUG信息 -->
        <logger name="org.springframework" level="INFO">
</logger>
        <logger name="org.mybatis" level="INFO"></logger>
        <root level="INFO">
            <appender-ref ref="Console" />
            <appender-ref ref="log" />
            <appender-ref ref="RollingFileInfo" />
            <appender-ref ref="RollingFileError" />
        </root>
    </loggers>

</configuration>

```

(3) 输出日志的代码示例

```

package jpm.logback;

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

public class TestLogback {

    public static void main(String[] args) {
        final Logger LOGGER =
LoggerFactory.getLogger(TestLogback.class);
        LOGGER.debug("print debug log.");
        LOGGER.info("print info log.");
        LOGGER.error("print error log.");
    }
}

```

(4) 打印日志结果

G:\jpm\logback\error\2018-05-20

包含到库中 共享 新建文件夹

名称

error-log.log

G:\jpm\logback\info\2018-05-20

包含到库中 共享 新建文件夹

名称

info-log.log

```
log.log - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[[2018-05-20 11:38:29.634] [INFO ] jpm.log4j2.TestLog4j2.main(TestLog4j2.java:11) - TestLog4j2 info log.
[2018-05-20 11:38:29.639] [ERROR] jpm.log4j2.TestLog4j2.main(TestLog4j2.java:12) - TestLog4j2 error log.

info.log - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[[2018-05-20 11:38:29.634] [INFO ] jpm.log4j2.TestLog4j2.main(TestLog4j2.java:11) - TestLog4j2 info log.

error.log - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[2018-05-20 11:38:29.639] [ERROR] jpm.log4j2.TestLog4j2.main(TestLog4j2.java:12) - TestLog4j2 error log.

[[2018-05-20 11:38:29.634] [INFO ] jpm.log4j2.TestLog4j2.main(TestLog4j2.java:11) - TestLog4j2 info log.
[2018-05-20 11:38:29.639] [ERROR] jpm.log4j2.TestLog4j2.main(TestLog4j2.java:12) - TestLog4j2 error log. console
```

上例使用 SLF4J 做日志接口，logback 做日志实现的日志示例

6. log4j2 使用示例

log4j2 介绍

Apache Log4j 2 是对 Log4j 的升级，与其前身 Log4j 1.x 相比有了显著的改进，并提供了许多 Logback 可用的改进，同时支持 JCL 以及 SLF4J。

log4j2 使用步骤说明

(1) 选择 jar 包

引入 log4j2 必要的包：log4j-api、log4j-core。

pom.xml 配置

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.8.2</version>
</dependency>

<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
```

```
<version>2.8.2</version>
</dependency>
```

(2) log2j 配置文件: log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 日志级别以及优先级排序: OFF > FATAL > ERROR > WARN > INFO >
DEBUG > TRACE > ALL -->
<!-- status用来指定log4j本身的打印日志的级别 -->
<!--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="WARN" monitorInterval="30">
  <!--先定义所有的appender -->
  <appenders>
    <!--这个输出控制台的配置 -->
    <console name="Console" target="SYSTEM_OUT">
      <!--输出日志的格式 -->
      <PatternLayout
        pattern="%d{yyyy-MM-dd HH:mm:ss.SSS}
[%-5level] %l %logger{36} - %msg%n" />
    </console>

    <!--定义输出到指定位置的文件 -->
    <File name="log" fileName="/jpm/log4j2/logs/log.log"
append="true">
      <PatternLayout
        pattern="%d{yyyy-MM-dd HH:mm:ss.SSS}
[%-5level] %l %logger{36} - %msg%n" />
    </File>

    <!-- 这个会打印出所有的info及以下级别的信息, 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 -->
    <RollingFile name="RollingFileInfo"
fileName="/jpm/log4j2/logs/info.log"
filePattern="/jpm/log4j2/logs/${date:yyyy-MM}/info-%d{yyyy-MM-dd}-%i.log">
      <!--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" />
      <PatternLayout
        pattern="%d{yyyy-MM-dd HH:mm:ss.SSS}
[%-5level] %l %logger{36} - %msg%n" />
    </RollingFile>
  </appenders>

  <policies>
    <TimeBasedTriggeringPolicy />
    <SizeBasedTriggeringPolicy size="100 MB" />
  </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30 -->
```

```

        <DefaultRolloverStrategy max="30" />
    </RollingFile>

    <RollingFile name="RollingFileError"
fileName="/jpm/log4j2/logs/error.log"
        filePattern="/jpm/log4j2/logs/${date:yyyy-MM}/error-%d{yyyy-MM-dd}-%i.log">
        <ThresholdFilter level="error" onMatch="ACCEPT"
onMismatch="DENY" />
        <PatternLayout
            pattern="[%d{yyyy-MM-dd HH:mm:ss.SSS}]
[%-5level] %l %logger{36} - %msg%n" />
        <Policies>
            <TimeBasedTriggeringPolicy />
            <SizeBasedTriggeringPolicy size="100 MB" />
        </Policies>
    </RollingFile>
</appenders>

<!--只有定义了logger并引入的appender， appender才会生效 -->
<loggers>
    <!--过滤掉spring和mybatis的一些无用的DEBUG信息 -->
    <logger name="org.springframework" level="INFO">
</logger>
    <logger name="org.mybatis" level="INFO"></logger>
    <root level="INFO">
        <appender-ref ref="Console" />
        <appender-ref ref="log" />
        <appender-ref ref="RollingFileInfo" />
        <appender-ref ref="RollingFileError" />
    </root>
</loggers>

</configuration>

```

(3) 输出日志的代码示例

```

package jpm.log4j2;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class TestLog4j2 {

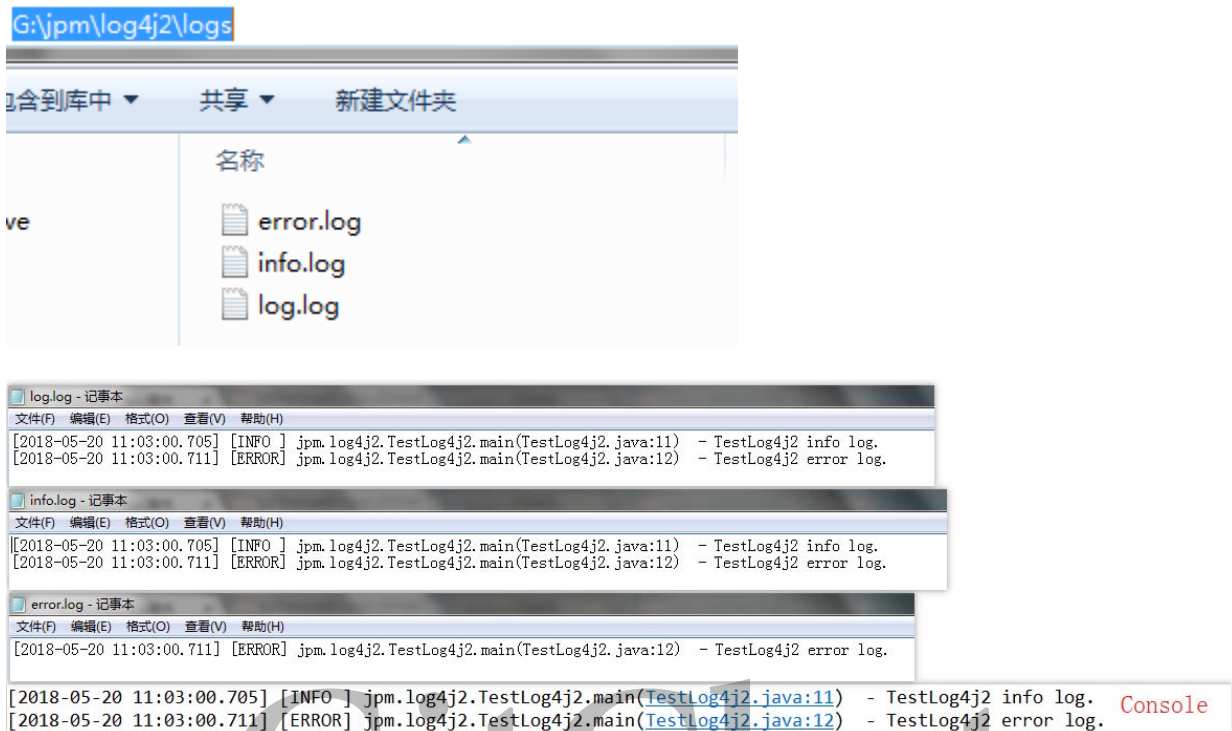
    public static void main(String[] args) {
        final Logger LOGGER =
LogManager.getLogger(LogManager.ROOT_LOGGER_NAME);
        LOGGER.debug("TestLog4j2 debug log.");
        LOGGER.info("TestLog4j2 info log.");
        LOGGER.error("TestLog4j2 error log.");
    }
}

```


}

}

(4) 打印日志结果



(5) 关于日志 level

共有 8 个级别，按照从低到高为：All < Trace < Debug < Info < Warn < Error < Fatal < OFF。

- **All**: 最低等级的，用于打开所有日志记录
- **Trace**: 是追踪，就是程序推进以下，你就可以写个 trace 输出，所以 trace 应该会特别多，不过没关系，我们可以设置最低日志级别不让他输出
- **Debug**: 指出细粒度信息事件对调试应用程序是非常有帮助的
- **Info**: 消息在粗粒度级别上突出强调应用程序的运行过程
- **Warn**: 输出警告及 warn 以下级别的日志
- **Error**: 输出错误信息日志
- **Fatal**: 输出每个严重的错误事件将会导致应用程序的退出的日志
- **OFF**: 最高等级的，用于关闭所有日志记录

程序会打印高于或等于所设置级别的日志，设置的日志等级越高，打印出来的日志就越少。

7. JCL(java common logging) + log4j 使用示例

JCL(java common logging) + log4j 介绍

使用 commons-logging 的 Log 接口，并由 commons-logging 在运行时决定使用哪种日志架构（如 Log4j）。现在，Apache 通用日志工具 commons-logging 和 Log4j 已经成为 Java

日志的标准工具，这个组合是比较常用的一个日志框架组合。

JCL(java common logging) + log4j 使用步骤说明

(1) 选择 jar 包

commons-logging-1.2 + log4j1.2.17

pom.xml

```
<dependency>
  <groupId>commons-logging</groupId>
  <artifactId>commons-logging</artifactId>
  <version>1.2</version>
</dependency>

<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
```

(2) 配置 common-logging.properties 文件

只需要一行即可，放在 classpath 下，如果是 Maven 中就在 src/resources 下，不过如果没有 common-logging.properties 文件，但是 src 下有 log4j.properties 配置也可以正常的输出 log4j 设置的日志。

```
org.apache.commons.logging.Log=org.apache.commons.logging.impl.Log4JLogger
```

(3) log4j.properties 配置

参考《log4j 使用步骤说明》

(4) 输出日志的代码示例

```
package jpm.jcllog4j;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class TestJclAndLog4j {

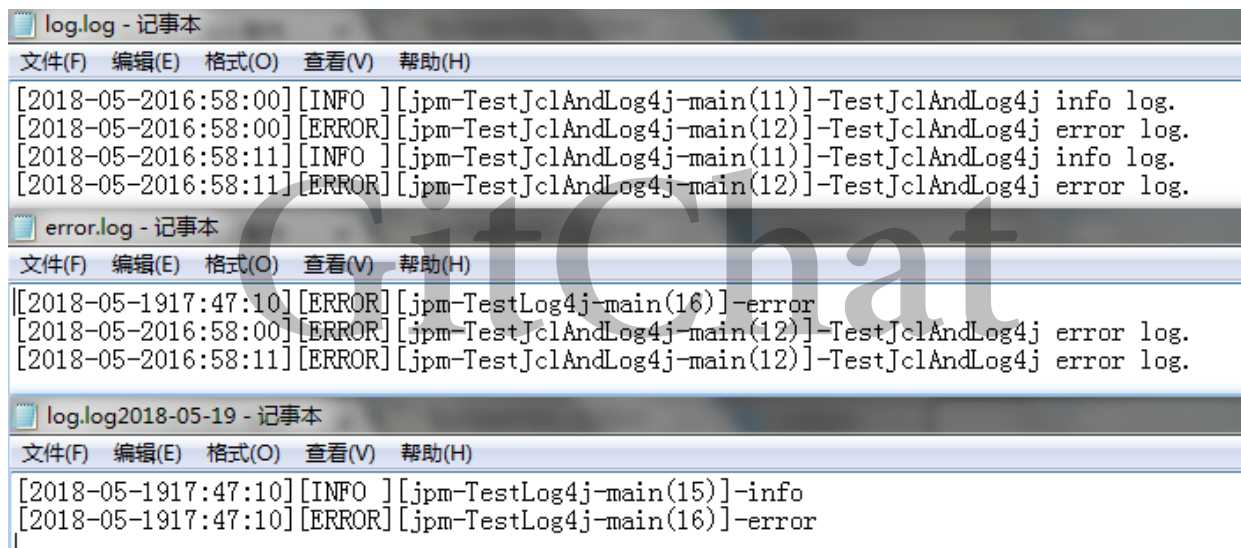
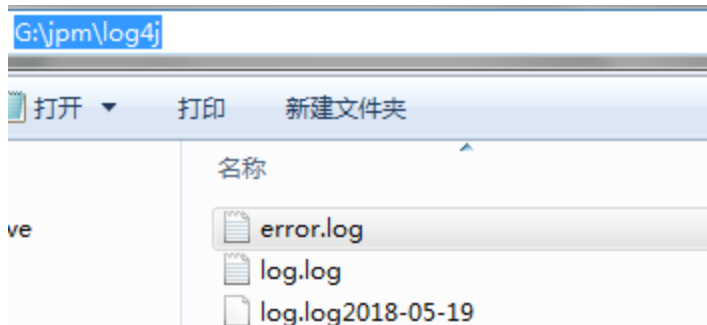
    public static void main(String[] args) {
```

```

        final Log LOGGER =
LogFactory.getLog(TestJclAndLog4j.class);
        LOGGER.debug("TestJclAndLog4j debug log.");
        LOGGER.info("TestJclAndLog4j info log.");
        LOGGER.error("TestJclAndLog4j error log.");
    }
}

```

(5) 打印日志结果



8. SLF4J + log4j 使用示例

SLF4J + log4j 介绍

SLF4j+Log4j 与 JCL+Log4J 的使用方式差不多，主要差异就在 SLF4J 用 用绑定包（slf4j-log4j12.jar）来告知用哪种日志实现，而 JCL 是通过配置文件来获得该选择哪个日志实现。

SLF4J + log4j使用步骤说明

(1) 选择jar包

slf4j-api.jar + slf4j-log4j12.jar

```

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.25</version>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.7.25</version>
</dependency>

```

(2) log4j.properties 配置文件

参考《log4j 使用步骤说明》

(3) 输出日志的代码示例

```

package jpm.slf4jlog4j;

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

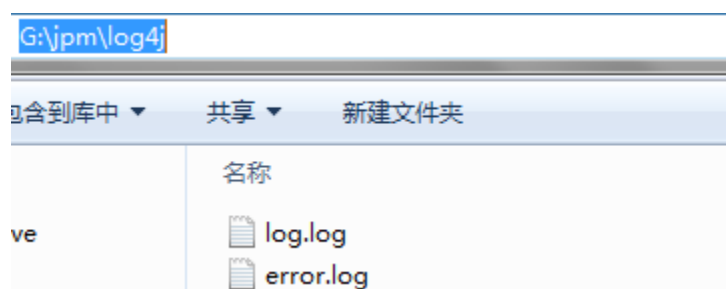
public class TestSlf4jAndLog4j {

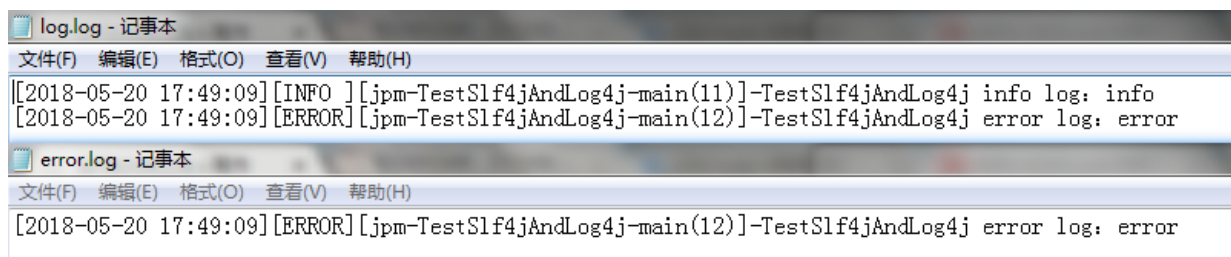
    public static void main(String[] args) {
        final Logger LOGGER =
        LoggerFactory.getLogger(TestSlf4jAndLog4j.class);
        LOGGER.debug("TestSlf4jAndLog4j debug log: {}",
"debug");
        LOGGER.info("TestSlf4jAndLog4j info log: {}",
"info");
        LOGGER.error("TestSlf4jAndLog4j error log: {}",
"error");
    }

}

```

(4) 打印日志结果





```
log.log - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[[2018-05-20 17:49:09][INFO][jpm-TestSlf4jAndLog4j-main(11)]-TestSlf4jAndLog4j info log: info
[2018-05-20 17:49:09][ERROR][jpm-TestSlf4jAndLog4j-main(12)]-TestSlf4jAndLog4j error log: error

error.log - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
[2018-05-20 17:49:09][ERROR][jpm-TestSlf4jAndLog4j-main(12)]-TestSlf4jAndLog4j error log: error
```

五、项目中日志记录遵守哪些原则

1. 阿里巴巴 Java 开发手册的日志规约

大家可以参考去年 10 月份阿里推出的《阿里巴巴 Java 开发手册（终极版）.pdf》里的第二章第二节“日志规约”部分的内容。

2. 个人经验总结

记录项目日志要注意以下几点：

1. 注意日志级别，尤其是 info 和 error 不能用混。
2. 注意记录信息的准确性，切记日志表达不清楚。
3. 注意不同的代码段日志说明不能重复。
4. 捕获异常后，要及时记录异常详细信息，并把异常传递到外部。
5. 时刻铭记，日志的记录是为了后期查询问题带来方便，因此重要的代码务必要记录日志。

六、一个推荐的项目 Java 日志实例

经过以上代码的分析，我们项目中使用的日志一般会选用一个日志接口和一个具体的日志实现。

那么日志接口是选 JCL 呢，还是选 SLF4J 呢？下面我们把他俩做个对比，具体日志实现为 log4j，如下图：

SLF4J+Log4j 组合	区别	JCL+Log4J 组合
slf4j-api-1.7.25.jar	接口 API	commons-logging-1.2.jar
slf4j-log4j12-1.7.25.jar	左边是用绑定包 右边是用配置文件来指定日志实现	commons-logging.properties，内容为： org.apache.commons.logging.LogFactory= org.apache.commons.logging.impl.LogFactoryImpl 或者 org.apache.commons.logging.Log= org.apache.commons.logging.impl.Log4JLogger
log4j-1.2.17.jar	一样	log4j-1.2.17.jar
log4j.properties	一样，原来怎么配置现在也怎么配	log4j.properties
程序代码中： import org.slf4j.Logger; import org.slf4j.LoggerFactory; Logger logger = LoggerFactory.getLogger(TestSlf4jAndLog4j); logger.info("Hello {}", "TestSlf4jAndLog4j");	左边引入的是 SLF4J API 右边引入的是 JCL 的 API SLF4J 支持参数化，而 JCL 不能	程序代码中： import org.apache.commons.logging.Log; import org.apache.commons.logging.LogFactory; Log log = LogFactory.getLog(TestJclAndLog4j.class); log.info("Hello TestJclAndLog4j");

从上图可以看出，用 slf4j 作为日志接口，对项目来说更好一些。

所以，在实际项目中，我一般建议使用 SLF4J+log4j 或者 slf4j+logback。

配置文件参见第四节

推荐程序片段如下

```
try {
    LOGGER.info("根据用户编码查询用户信息-开始，userId:
{}" , userId);
    User user = userService.getUserById(userId);
    LOGGER.info("根据用户编码查询用户信息-结束，userId:
{}" , userId);
} catch (CustomException e) {
    LOGGER.error("根据用户编码查询用户信息-自定义异常:{}"
, e.getMessage());
    throw new CustomException("根据用户编码查询用户信息-
自定义异常{}" , e.getMessage(), e);
} catch (Exception e) {
    LOGGER.error("根据用户编码查询用户信息-捕获异常:{}" ,
e.toString());
    throw new ServiceException(根据用户编码查询用户信息-
捕获异常:{}" , e.toString(), e);
}
```

到此，Java 日志从入门到实战 Chat 就结束了，咱们下次再见，谢谢。