# 日志文件

日志文件是用于记录系统操作事件的文件集合。

日志文件它具有处理历史数据、诊断问题的追踪以及理解系统的活动等重要的作用。

## 调试日志

在软件开发中，我们要去经常的调试程序，或者做一些状态的输出，便于我们查询程序的运行状况。为了让我们能够更加灵活且方便的控制这些调试信息，我们肯定是需要更加专业的日志技术。我们平时在调试程序的过程中所使用的肯定就是专业开发工具自带的debug功能，可以实时查看程序运行情况，不能够有效保存运行情况的信息。调试日志是能够更加方便的去“重现”这些问题。

## 系统日志

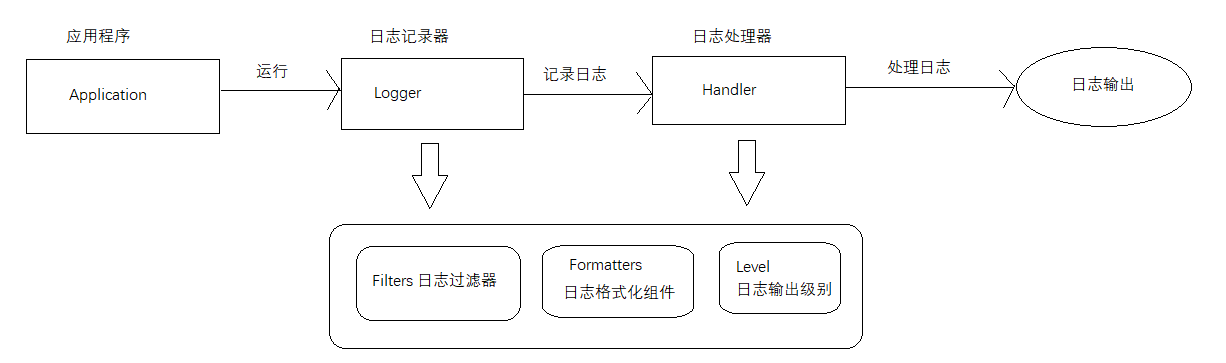
系统日志是用来记录系统中硬件、软件和系统相关问题的信息。同时还可以监视系统中发生的事件。用户可以通过它来检查错误发生的原因，或者寻找收到攻击是留下的痕迹。

系统日志包括系统日志、应用日志和安全日志这几种分类

# JUL简介

JUL全程 Java Util Logging，它是java原生的日志框架，使用时不需要另外引用第三方的类库，相对其他的框架使用方便，学习简单，主要是使用在小型应用中。

## JUL组件介绍



Logger：被称为记录器，应用程序通过获取Logger对象，抵用其API来发布日志信息。Logger通常被认为是访问日志系统的入口程序。

Handler：处理器，每个Logger都会关联一个或者是一组Handler，Logger会将日志交给关联的Handler去做处理，由Handler负责将日志做记录。Handler具体实现了日志的输出位置，比如可以输出到控制台或者是文件中等等。

Filter：过滤器，根据需要定制哪些信息会被记录，哪些信息会被略过。

Formatter：格式化组件，它负责对日志中的数据和信息进行转换和格式化，所以它决定了我们输出日志最终的形式。

Level：日志的输出级别，每条日志消息都有一个关联的级别。我们根据输出级别的设置，用来展现最终所呈现的日志信息。根据不同的需求，去设置不同的级别。

# Log4j简介

Log4j是[Apache](https://baike.baidu.com/item/Apache/8512995" \t "_blank)的一个开源项目，通过使用Log4j，我们可以控制日志信息输送的目的地是[控制台](https://baike.baidu.com/item/%E6%8E%A7%E5%88%B6%E5%8F%B0/2438626" \t "_blank)、文件、[GUI](https://baike.baidu.com/item/GUI" \t "_blank)组件，甚至是套接口服务器、[NT](https://baike.baidu.com/item/NT/3443842" \t "_blank)的事件记录器、[UNIX](https://baike.baidu.com/item/UNIX" \t "_blank) [Syslog](https://baike.baidu.com/item/Syslog" \t "_blank)[守护进程](https://baike.baidu.com/item/%E5%AE%88%E6%8A%A4%E8%BF%9B%E7%A8%8B/966835" \t "_blank)等；我们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。最令人感兴趣的就是，这些可以通过一个[配置文件](https://baike.baidu.com/item/%E9%85%8D%E7%BD%AE%E6%96%87%E4%BB%B6/286550" \t "_blank)来灵活地进行配置，而不需要修改应用的代码。

官方网站： <http://logging.apache.org/log4j/1.2/> Log for java

我们使用log4j技术，主要使用的是其配置文件

## Log4j组件介绍

Log4j主要由 Loggers (日志记录器)、Appenders（输出控制器）和 Layout（日志格式化器）组成。其中 Loggers 控制日志的输出以及输出级别(JUL做日志级别Level)；Appenders 指定日志的输出方式（输出到控制台、文件等）；Layout 控制日志信息的输出格式。

### Loggers

关于日志级别信息，例如DEBUG、INFO、WARN、ERROR…级别是分大小的，DEBUG < INFO < WARN < ERROR，分别用来指定这条日志信息的重要程度，Log4j输出日志的规则是：只输出级别不低于设定级别的日志信息，假设Loggers级别设定为INFO，则INFO、WARN、ERROR级别的日志信息都会输出，而级别比INFO低的DEBUG则不会输出。

### Loggers**Appenders**

记录日志以及定义日志的级别仅仅是Log4j的基本功能，Log4j日志系统还提供许多强大的功能，比如允许把日志输出到不同的地方，如控制台（Console）、文件（Files）等，可以根据天数或者文件大小产生新的文件，可以以流的形式发送到其它地方等等。

常用Appenders：

ConsoleAppender

将日志输出到控制台

FileAppender

将日志输出到文件中

DailyRollingFileAppender

将日志输出到一个日志文件，并且每天输出到一个新的文件

RollingFileAppender

将日志信息输出到一个日志文件，并且指定文件的尺寸，当文件大小达到指定尺寸时，会自动把文件改名，同时产生一个新的文件

JDBCAppender

把日志信息保存到数据库中

### Layouts

有时用户希望根据自己的喜好格式化自己的日志输出，Log4j可以在Appenders的后面附加Layouts来完成这个功能。Layouts提供四种日志输出样式，如根据HTML样式、自由指定样式、包含日志级别与信息的样式和包含日志时间、线程、类别等信息的样式。

常用Layouts:

HTMLLayout

格式化日志输出为HTML表格形式

SimpleLayout

简单的日志输出格式化，打印的日志格式如默认INFO级别的消息

PatternLayout

最强大的格式化组件，可以根据自定义格式输出日志，如果没有指定转换格式， 就是用默认的转换格式

## 日志输出格式说明

使用PatternLayout可以自定义格式输出，是我们最常用的方式

这种格式化输出采用类似于 C 语言的 printf 函数的打印格式格式化日志信息，具体的占位符及其含义如下：

%m 输出代码中指定的日志信息

%p 输出优先级，及 DEBUG、INFO 等

%n 换行符（Windows平台的换行符为 "\n"，Unix 平台为 "\n"）

%r 输出自应用启动到输出该 log 信息耗费的毫秒数

%c 输出打印语句所属的类的全名

%t 输出产生该日志的线程全名

%d 输出服务器当前时间，默认为 ISO8601，也可以指定格式，如：%d{yyyy年MM月dd日 HH:mm:ss}

%l 输出日志时间发生的位置，包括类名、线程、及在代码中的行数。如：Test.main(Test.java:10)

%F 输出日志消息产生时所在的文件名称

%L 输出代码中的行号

%% 输出一个 "%" 字符

可以在 % 与字符之间加上修饰符来控制最小宽度、最大宽度和文本的对其方式。如：

%5c 输出category名称，最小宽度是5，category<5，默认的情况下右对齐

%-5c 输出category名称，最小宽度是5，category<5，"-"号指定左对齐,会有空格

%.5c 输出category名称，最大宽度是5，category>5，就会将左边多出的字符截掉，<5不

会有空格

%20.30c category名称<20补空格，并且右对齐，>30字符，就从左边交远销出的字符截掉

# JCL

全称为Jakarta Commons Logging，是Apache提供的一个**通用日志API**。

用户可以自由选择第三方的日志组件作为具体实现，像log4j，或者jdk自带的jul， common-logging会通过动态查找的机制，在程序运行时自动找出真正使用的日志库。

当然，common-logging内部有一个Simple logger的简单实现，但是功能很弱。所以使用common-logging，通常都是配合着log4j以及其他日志框架来使用。

使用它的好处就是，代码依赖是common-logging而非log4j的API， 避免了和具体的日志API直接耦合，在有必要时，可以更改日志实现的第三方库。

JCL 有两个基本的抽象类：

Log：日志记录器

LogFactory：日志工厂（负责创建Log实例）

# SLF4J

简单日志门面(Simple Logging Facade For Java) SLF4J主要是为了给Java日志访问提供一套标准、规范的API框架，其主要意义在于提供接口，具体的实现可以交由其他日志框架，例如log4j和logback等。 当然slf4j自己也提供了功能较为简单的实现，但是一般很少用到。对于一般的Java项目而言，日志框架会选择slf4j-api作为门面，配上具体的实现框架（log4j、logback等），中间使用桥接器完成桥接。所以我们可以得出SLF4J最重要的两个功能就是对于日志框架的绑定以及日志框架的桥接。

官方网站： <https://www.slf4j.org/>

前面介绍的几种日志框架，每一种日志框架都有自己单独的API，要使用对应的框架就要使用其对应的API，这就大大的增加应用程序代码对于日志框架的耦合性。

为了解决这个问题，就是在日志框架和应用程序之间架设一个沟通的桥梁，对于应用程序来说，无论底层的日志框架如何变，都不需要有任何感知。只要门面服务做的足够好，随意换另外一个日志框架，应用程序不需要修改任意一行代码，就可以直接上线。

## 常见的日志框架及日志门面

常见的日志实现：JUL、log4j、logback、log4j2

常见的日志门面 ：JCL、slf4j

出现顺序 ：log4j -->JUL-->JCL--> slf4j --> logback --> log4j2

## SLFJ桥接技术

通常，我们依赖的某些组件依赖于SLF4J以外的日志API。我们可能还假设这些组件在不久的将来不会切换到SLF4J。为了处理这种情况，SLF4J附带了几个桥接模块，这些模块会将对log4j，JCL和java.util.logging API的调用重定向为行为，就好像是对SLF4J API进行的操作一样。

# Logback

Logback是由[log4j](https://baike.baidu.com/item/log4j" \t "_blank)创始人设计的又一个开源日志组件。

Logback当前分成三个模块：logback-core,logback- classic和logback-access。

logback-core是其它两个模块的基础模块。

logback-classic是log4j的一个改良版本。此外logback-classic完整实现SLF4J API。使你可以很方便地更换成其它日志系统如log4j或JDK14 Logging。

logback-access访问模块与[Servlet](https://baike.baidu.com/item/Servlet/477555" \t "_blank)容器集成提供通过Http来访问日志的功能。

## Logback中的组件

Logger: 日志的记录器，主要用于存放日志对象，也可以定义日志类型、级别。

Appender:用于指定日志输出的目的地，目的地可以是控制台、文件、数据库等等。

Layout: 负责把事件转换成字符串，格式化的日志信息的输出。

在Logback中Layout对象被封装在encoder中。

也就是说我们未来使用的encoder其实就是Layout

## Logback配置文件

Logback提供了3种配置文件

logback.groovy

logback-test.xml

logback.xml

如果都不存在则采用默认的配置

## 日志输出格式

日志输出格式：  
 %-10level 级别 案例为设置10个字符，左对齐  
 %d{yyyy-MM-dd HH:mm:ss.SSS} 日期  
 %c 当前类全限定名  
 %M 当前执行日志的方法  
 %L 行号  
 %thread 线程名称  
 %m或者%msg 信息  
 %n 换行

# Log4j2

Apache Log4j 2是对Log4j的升级，它比其前身Log4j 1.x提供了重大改进，并提供了Logback中可用的许多改进，同时修复了Logback架构中的一些问题。被誉为是目前最优秀的Java日志框架。

## Log4j2特征

**性能提升**

Log4j2包含基于LMAX Disruptor库的下一代异步记录器。在多线程场景中，异步记录器的吞吐量比Log4j 1.x和Logback高18倍，延迟低。

**自动重新加载配置**

与Logback一样，Log4j2可以在修改时自动重新加载其配置。与Logback不同，它会在重新配置发生时不会丢失日志事件。

**高级过滤**

与Logback一样，Log4j2支持基于Log事件中的上下文数据，标记，正则表达式和其他组件进行过滤。

此外，过滤器还可以与记录器关联。与Logback不同，Log4j2可以在任何这些情况下使用通用的Filter类。

**插件架构**

Log4j使用插件模式配置组件。因此，您无需编写代码来创建和配置Appender，Layout，Pattern Converter等。在配置了的情况下，Log4j自动识别插件并使用它们。

**无垃圾机制**

在稳态日志记录期间，Log4j2 在独立应用程序中是无垃圾的，在Web应用程序中是低垃圾。这减少了垃圾收集器的压力，并且可以提供更好的响应性能。

目前市面上最主流的日志门面就是SLF4J，虽然Log4j2 也是日志门面，因为它的日志实现功能非常强大，性能优越。所以我们一般情况下还是将 Log4j2 看作是日志的实现

SLF4j + Log4j2 的组合，是市场上最强大的日志功能实现方式，绝对是未来的主流趋势。

## 异步日志

异步日志是log4j2最大的特色，其性能的提升主要也是从异步日志中受益。

Log4j2提供了两种实现日志的方式，一个是通过AsyncAppender，一个是通过AsyncLogger，分别对应前面我们说的Appender组件和Logger组件。

注意这是两种不同的实现方式，在设计和源码上都是不同的体现。

**AsyncAppender方式**

是通过引用别的Appender来实现的，当有日志事件到达时，会开启另外一个线程来处理它们。需要注意的是，如果在Appender的时候出现异常，对应用来说是无法感知的。 AsyncAppender应该在它引用的Appender之后配置，默认使用 java.util.concurrent.ArrayBlockingQueue实现而不需要其它外部的类库。 当使用此Appender的时候，在多线程的环境下需要注意，阻塞队列容易受到锁争用的影响，这可能会对性能产生影响。这时候，我们应该考虑使用无锁的异步记录器（AsyncLogger）。

**AsyncLogger方式**

AsyncLogger才是log4j2实现异步最重要的功能体现，也是官方推荐的异步方式。

它可以使得调用Logger.log返回的更快。你可以有两种选择：全局异步和混合异步。

全局异步：所有的日志都异步的记录，在配置文件上不用做任何改动，只需要在jvm启动的时候增加一个参数即可实现。

混合异步：你可以在应用中同时使用同步日志和异步日志，这使得日志的配置方式更加灵活。虽然Log4j2提供以一套异常处理机制，可以覆盖大部分的状态，但是还是会有一小部分的特殊情况是无法完全处理的，比如我们如果是记录审计日志（特殊情况之一），那么官方就推荐使用同步日志的方式，而对于其他的一些仅仅是记录一个程序日志的地方，使用异步日志将大幅提升性能，减少对应用本身的影响。

混合异步的方式需要通过修改配置文件来实现，使用AsyncLogger标记配置。

性能对比：

