Java日志处理

# 简介

日志系统是一种不可或缺的跟踪调试工具，特别是在任何无人职守的后台程序以及那些没有跟踪调试环境的系统中有着广泛的应用。长期以来，日志系统作为一种应用程序服务，对于跟踪调试、程序状态记录、崩溃数据恢复都有非常现实的意义。这种服务通常以两种方式存在：

* 1.日志系统作为服务进程存在。Windows中的的事件日志服务就属于这种类型，该类型的日志系统通常通过消息队列机制将所需要记录的日志由日志发送端 发送给日志服务。日志发送端和日志保存端通常不在同一进程当中，日志的发送是异步过程。这种日志服务通常用于管理员监控各种系统服务的状态。
* 2.日志系统作为系统调用存在。Java世界中的日志系统和[Unix](http://www.chinabyte.com/keyword/unix/)环境下诸多守护进程所使用的日志系统都属于这种类型。日志系统的代码作为系统调用被编译进日志发送端，日志系统的运行和业务代码的运行在同一进程空间。日志的发送多数属于同步过程。这种日志服务由于能够同步反映处系统运行状态，通常用于调试跟踪和崩溃恢复。

## 误区

　在控制台环境上调试Java程序时，此时往控制台或者文本文件输出一段文字是查看程序运行状态最简单的做法，但这种方式并不能解决全部的问题。有时候， 对于一个我们无法实时查看系统输出的系统或者一个确实需要保留我们输出信息的系统，良好的日志系统显得相当必要。因此，不能随意的输出各种不规范的调试信 息，这些随意输出的信息是不可控的，难以清除，可能为后台监控、错误排除和错误恢复带来相当大的阻力。

## 日志框架的基本功能

一个完备的日志系统框架通常应当包括如下基本特性：

　　所输出的日志拥有自己的分类：这样在调试时便于针对不同系统的不同[模块](http://www.chinabyte.com/keyword/%E6%A8%A1%E5%9D%97/)进行查询，从而快速定位到发生日志事件的代码。

　　日志按照某种标准分成不同级别：分级以后的日志，可以用于同一分类下的日志筛选。

　　支持多线程：日志系统通常会在多线程环境中使用，特别是在Java系统当中，因此作为一种系统资源，日志系统应当保证是线程安全的。

　　支持不同的记录媒介：不同的工程项目往往对日志系统的记录媒介要求不同，因此日志系统必须提供必要的开发接口，以保证能够比较容易的更换记录介质。

　　高性能：日志系统通常要提供高速的日志记录功能以应对大系统下大请求流量下系统的正常运转。

　　稳定性：日志系统必须是保持高度的稳定性，不能因为日志系统内部错误导致主要业务代码的崩溃。

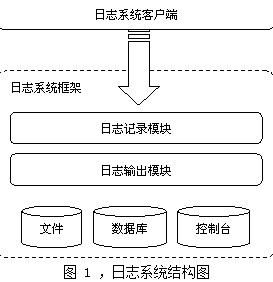
## 常用日志系统简介

在Java世界中，以下三种日志框架比较优秀：

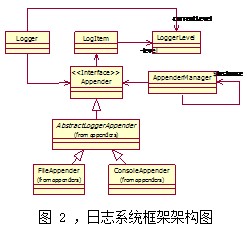
* Log4J最早的Java日志框架之一，由Apache基金会发起，提供灵活而强大的日志记录机制。但是其复杂的配置过程和内部概念往往令使用者望而却步。
* JDK1.4LoggingFramework该框架同样是Apache基金会项目，其出现主要是为了使得Java项目能够在Log4J和JDK1.4lLoggingFramework的使用上随意进行切换，因此该框架提供了统一的调用接口和配置方法
* CommonsLoggingFramwork 该框架同样是Apache基金会项目，其出现主要是为了使得Java项目能够在Log4J和JDK1.4lLoggingFramework的使用上随意进行切换，因此该框架提供了统一的调用接口和配置方法。

# 系统架构

日志系统框架可以分为日志记录模块和日志输出模块两大部分。日志记录模块负责创建和管理日志记录器(Logger)，每一个Logger对象负责按照不同 的级别(LoggerLevel)接收各种记录了日志信息的日志对象(LogItem)，Logger对象首先获取所有需要记录的日志，并且同步地将日志 分派给日志输出模块。日志输出模块则负责日志输出器(Appender)的创建和管理，以及日志的输出。系统中允许有多个不同的日志输出器，日志输出器负责将日志记录到[存储介质](http://www.chinabyte.com/keyword/%E5%AD%98%E5%82%A8%E4%BB%8B%E8%B4%A8/)当中。



## UML



日志记录器Logger是整个日志系统框架的用户使用接口，程序员可以通过该接口记录日志，为了实现对日志进行分类，系统设计允许存在多个Logger对 象，每一个Logger负责一类日志的记录，Logger类同时实现了对其对象本身的管理。LoggerLevel类定义了整个日志系统的级别，在客户端 创建和发送日志时，这些级别会被使用到。Logger对象在接收到客户端创建和发送的日志消息时，同时将该日志消息包装成日志系统内部所使用的日志对象 LogItem，日志对象除了发送端所发送的消息以外，还会包装诸如发送端类名、发送事件、发送方法名、发送行号等等。这些额外的消息对于系统的跟踪和调 试都非常有价值。包装好的LogItem最终被发送给输出器，由这些输出器负责将日志信息写入最终媒介，输出器的类型和个数均不固定，所有的输出器通过 AppenderManager进行管理，通常通过配置文件即可方便扩展出多个输出器。

# LogBack

# Log4J 1.x

## Why logging?

Inserting log statements into your code is a low-tech method for debugging it. It may also be the only way because debuggers are not always available or applicable. This is often the case for distributed applications.

On the other hand, some people argue that log statements pollute source code and decrease legibility. (We believe that the contrary is true). In the Java language where a preprocessor is not available, log statements increase the size of the code and reduce its speed, even when logging is turned off. Given that a reasonably sized application may contain thousands of log statements, speed is of particular importance.

### Why log4j?

With log4j it is possible to enable logging at runtime without modifying the application binary. The log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost. Logging behavior can be controlled by editing a configuration file, without touching the application binary.

Logging equips the developer with detailed context for application failures. On the other hand, testing provides quality assurance and confidence in the application. Logging and testing should not be confused. They are complementary. When logging is wisely used, it can prove to be an essential tool.

One of the distinctive features of log4j is the notion of inheritance in loggers. Using a logger hierarchy it is possible to control which log statements are output at arbitrarily fine granularity but also great ease. This helps to reduce the volume of logged output and the cost of logging.

The target of the log output can be a file, an OutputStream, a java.io.Writer, a remote log4j server, a remote Unix Syslog daemon, or many other output targets.

### Performance

On an AMD Duron clocked at 800Mhz running JDK 1.3.1, it costs about 5 nanoseconds to determine if a logging statement should be logged or not. Actual logging is also quite fast, ranging from 21 microseconds using the SimpleLayout, 37 microseconds using the TTCCLayout. The performance of the PatternLayout is almost as good as the dedicated layouts, except that it is much more flexible.

## Loggers, Appenders and Layouts

Log4j has three main components: *loggers*, *appenders* and *layouts*. These three types of components work together to enable developers to log messages according to message type and level, and to control at runtime how these messages are formatted and where they are reported.

### Logger hierarchy

The first and foremost advantage of any logging API over plain System.out.println resides in its ability to disable certain log statements while allowing others to print unhindered. This capability assumes that the logging space, that is, the space of all possible logging statements, is categorized according to some developer-chosen criteria. This observation had previously led us to choose *category* as the central concept of the package. However, since log4j version 1.2, Logger class has replaced the Category class. For those familiar with earlier versions of log4j, the Logger class can be considered as a mere alias to the Category class.

Loggers are named entities. Logger names are case-sensitive and they follow the hierarchical naming rule:

### Named Hierarchy

A logger is said to be an *ancestor* of another logger if its name followed by a dot is a prefix of the *descendant* logger name. A logger is said to be a *parent* of a *child* logger if there are no ancestors between itself and the descendant logger.