日志系统概述

参考：https://blog.csdn.net/romandion/article/details/1877189

日志系统的重要性

日志系统在整个系统架构中的重要性可以称得上基础的基础，但是这一点，都容易被大多数人所忽视。因为日志在很多人看来只是printf。在系统运行期间，是很难step by step的，所以只能根据系统的运行轨迹来推断错误出现的位置，这往往也是唯一的资料，特别是在高可靠性的情况下。

从更大方面的范围来说，日志系统是运营维护的范畴。但小的方面来说，这是必须的调试的手段。从多年的开发经验来看，日志系统是必须被我们重视的。

日志系统要解决的问题

日志系统的主要解决的问题是记录系统的运行轨迹，在这个基础上，进行跟踪分析错误，审计系统运行流程。在高可靠的系统中，是不允许系统运行终止的。

日志系统的内容可以分为两类，

> 一类可是业务级别的日志，主要供终端用户来分析他们业务过程；

> 另一类是系统级别的日志，供开发者维护系统的稳定。

由于日志系统的数据输出量比较大，所以不能不考虑对整个系统性能的影响。

从另外一方面来看，海量的日志内容有时候并不件好事，因为，很容易覆盖真实问题的蛛丝马迹,也增加日志阅读者信息检索的困难。

日志系统构成

一个日志系统根据他的过程，可以分为日志来源，系统控制，日志输出，日志存储。根据这个过程，我们可以将整个系统分为4个模块，并加以抽象。

> 1 日志来源： 日志内容可以来源于任何其他系统，但对日志系统来说，这是个格式化的缓冲区。对于日志系统来说，任何内容都是合法的。最重要的是，日志系统必须提供一个简单的规则，为后续的管理和检索提供方便性和灵活性。在传统的printf格式中，是很难维护一个格式化的日志输出。文本方式对人来说阅读方便，但不容易检索，特别是在大量日志的情况下，更不好维护了。

> 2 日志控制： 系统控制的重点在于控制日志内容在日志系统中的流转过程。比如日志输出目的地，比如日志的输出级别。我们在apache的Logging项目中曾经看到，他们提供了一个和平时不太一样的输出目的地，telnet。这和传统的stdout、stderr、syslog有很大的区别，便于远程管理，更大的潜力在于，可以在运行期，通过登录telnet来动态调整系统环境配置。

> 3 日志输出: 日志在控制台输出是比较常见的，但如何考虑为系统的可靠性提供支持，以及大量日志内容的情况下，这个一般不予考虑的。在控制台输出的，只会是非常核心的内容或者是致命的错误，况且，在有些情况下，不一样会有控制台。我们一般在这种情况下，都倾向于将日志输出到文件。但对一个完善的日志系统，日志输出和日志存储又是有区别的。日志存储是日志输出到文件的一种方式。日志输出也是日志控制的一个内容。

> 4 日志存储: 日志存储在很多小型系统往往并不需要关注，一个可靠性要求很高的系统中，对日志存储却是极为苛刻。就是在现在的数据库系统中，也必须依赖日志的存在，来还原操作。

日志系统设计思路

参考：https://blog.csdn.net/s_lisheng/article/details/79654542

日志库的设计，一般而言要抓住最核心的一条，就是**日志从产生到到达最终目的地期间的处理流程。**一般而言，为了设计一个灵活可扩展，可配置的日志库，主要将日志库分为4个部分去设计，分别是：记录器、过滤器、格式化器、输出器四部分。

> **记录器**：负责产生日志记录的原始信息，比如（原始信息，日志等级，时间，记录的位置）等等信息。

> **过滤器**：负责按指定的过滤条件过滤掉我们不需要的日志（比如按日志等级过滤）。

> **格式化器**：负责对原始日志信息按照我们想要的格式去格式化。

> **输出器**：负责将将要进行记录的日志（一般经过过滤器及格式化器的处理后）记录到日志目的地（例如：输出到文件中）

下面以一条日志的生命周为例说明日志库是怎么工作的。 一条日志的生命周期：

1. 产生。info(“log information.”);
2. 经过记录器。记录器去获取日志发生的时间，位置，线程信息等等信息，会有一个数据结构去存储你需要的信息（例如：msg:”log information.”,time:2018-3-20 10:00:00,level:info,location:main.rs:3 lines）
3. 经过过滤器。决定是否记录（例如，过滤条件设为info级以下的过滤掉，这里条日志信息等级是info，满足条件，继续。）
4. 经过格式化器。假设我们想输出为“2018-3-22 10:00:00 [info] log information.”

5. 到输出器。例如输出到文件中，我们就将这条信息写到文件上（File::write(....);）.
6. 这条日志信息生命结束了。

日志分类

syslog日志级别分类

在LINUX的SYSLOG中，对日志内容进行分级，将分为8个级别，如下：

```
#define LOG_EMERG   0   /* system is unusable */
#define LOG_ALERT   1   /* action must be taken immediately */
#define LOG_CRIT    2   /* critical conditions */
#define LOG_ERR     3   /* error conditions */
#define LOG_WARNING 4   /* warning conditions */
#define LOG_NOTICE  5   /* normal but significant condition */
#define LOG_INFO    6   /* informational */
#define LOG_DEBUG   7   /* debug-level messages */
```

> LOG_EMERG 系统不可用

> LOG_ALERT 消息需立即处理

> LOG_CRIT 重要情况

> LOG_ERR 错误

> LOG_WARNING 警告

> LOG_NOTICE 正常情况，但较为重要

> LOG_INFO 信息

> LOG_DEBUG 调试信息

syslog日志来源分类

```
#define LOG_KERN    (0<<3)  /* kernel messages */
#define LOG_USER    (1<<3)  /* random user-level messages */
#define LOG_MAIL    (2<<3)  /* mail system */
#define LOG_DAEMON  (3<<3)  /* system daemons */
#define LOG_AUTH    (4<<3)  /* security/authorization
messages */
#define LOG_SYSLOG  (5<<3)  /* messages generated internally
by syslogd */
#define LOG_LPR     (6<<3)  /* line printer subsystem */
#define LOG_NEWS    (7<<3)  /* network news subsystem */
#define LOG_UUCP    (8<<3)  /* UUCP subsystem */
#define LOG_CRON    (9<<3)  /* clock daemon */
#define LOG_AUTHPRIV    (10<<3) /* security/authorization
messages (private) */
#define LOG_FTP     (11<<3) /* ftp daemon */
```

boost.log库的设计

boost::log 的设计主要由日志器（Logger）、日志核心（Logging core）、 Sink前端（frontend）、Sink后端(backend）组成。

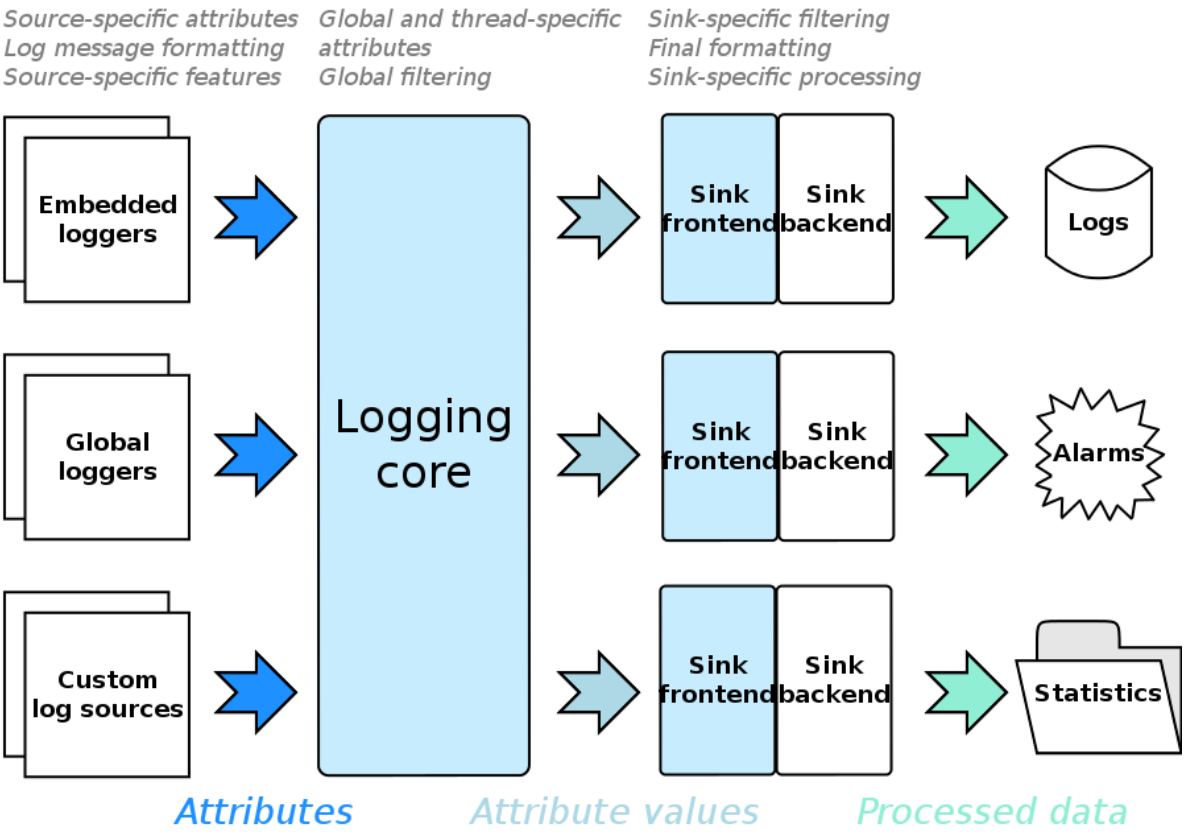
> 日志文本以及日志环境由日志器（Logger）负责搜集。

> 日志核心负责处理日志数据。例如全局过滤、将日志记录传递给Sink.

> Sink前端分为同步、异步以及不考虑线程同步问题的版本，它们负责将日志记录传递给 Sink后端处理。

> Sink 后端负责把日志记录格式化并输出到不同的介质中（例如日志文件、报警以及统计源中）。

架构图



boost.log相关概念

日志记录：一个独立的消息包，这个消息包还不是实际写到日志里的消息，它只是一个候选的消息。

属性：日志记录中的一个消息片。

属性值：那就是上面所说的属性的值了，可以是各种数据类型。

日志槽（LOG SINK)：日志写向的目标，它要定义日志被写向什么地方，以及如何写。

日志源：应用程序写日志时的入口，其实质是一个logger对象的实例。

日志过滤器：决定日志记录是否要被记录的一组判断。

日志格式化：决定日志记录输出的实际格式。

日志核心：维护者日志源、日志槽、日志过滤器等之间的关系的一个全局中的实体。主要在初始化logging library时用到。