Tina Linux syslog

使用指南

版本号: 1.2

发布日期: 2021.04.09

版本历史

版本号	日期	制/修订人	内容描述
1.0	2019.05.21	AWA1402	初始版本
1.1	2020.07.22	AWA1402	修改部分样式
1.2	2021.04.09	AWA1611	迁移至新仓库并新增图片

目　录

1 基本介绍

syslog 是一套统一管理系统日志的机制，尤其常用于记录守护进程的输出信息上。因为守护进程不存在控制终端，它的打印不能简单地直接输出到 stdin 或 stderr。

使用 syslog 时，一般需要关注两部分：**syslog 守护进程**与 **syslog 函数**。

1.1 syslog 守护进程

syslog 守护进程用于统一管理日志。它一般会创建一个数据报（SOCK_DGRAM）类型的 Unix 域套接字（Unix domain socket），将其捆绑到 /dev/log（不同的系统可能会有所不同）。如果支持网络功能，它可能还会创建一个 UDP 套接字，并捆绑到端口 514。syslog 守护进程从这些套接字中读取日志信息，然后再输出到设定的目标位置（文件、串口等）。

后面提到的 ubox 的 logd、busybox 的 syslogd、syslog-ng 都是 syslog 守护进程的不同实现。

1.2 syslog 函数

应用程序若想将打印信息发送到 syslog 守护进程，就需要通过 Unix 域套接字将信息输出到 syslog 守护进程绑定的路径，标准的做法是通过调用 syslog 函数：

```
#include <syslog.h>

void openlog(const char *ident, int option, int facility);
void syslog(int priority, const char *format, ...);
void closelog(void);
```

syslog 函数被应用程序首次调用时，会创建一个 Unix 域套接字，并连接到 syslog 守护进程的 Unix 域套接字绑定的路径名上。这个套接字会一直保持打开，直到进程终止为止。应用程序也可以显式地调用 openlog 和 closelog（这两个函数都不是必须要调用的），如果不显式调用，在第一次调用 syslog 函数时会自动隐式地调用 openlog ，进程结束后也会自动关闭与 syslog 守护进程通信的文件描述符，相当于隐式调用 closelog。

以下是一些参数说明，更详细的请参考 syslog 的 man 手册。

1.2.1 openlog()

- ident 参数会被添加到每一条日志信息中，一般为程序的名字。
- option 参数支持以下的值，可通过或操作（OR）让其支持多个 option：

表 1-1: openlog() 的 option 参数

option	说明
LOG_CONS	若日志无法通过 Unix 域套接字送到 syslog 守护进程，则将其输出到 console
LOG_NDELAY	立即打开至 syslog 守护进程 Unix 域套接字的连接，不要等到第一次调用 syslog 函数时才建立连接（通常情况下会在第一次调用 syslog 时才建立连接）
LOG_NOWAIT	不要等待在将消息计入日志过程中可能已经创建的子进程（GNU C 库中不会创建子进程，因此该选项在 Linux 中不会起作用）
LOG_ODELAY	LOG_NDELAY 的相反，在第一次调用 syslog 函数前不建立连接（这是默认的行为，可以不显式指定该选项）
LOG_PERROR	将日志信息也输出到 stderr
LOG_PID	在每条日志信息中添加上进程 ID

- facility 参数用于指定当前应用程序的设施类型，为后续的 syslog 调用指定一个设施的默认值。该参数的存在意义是让 syslog 守护进程可以通过配置文件对不同设施类型的日志信息做区分处理。如果应用程序没有调用 openlog，或是调用时 facility 参数为 0，可在调用 syslog 时将 facility 作为 priority 参数的一部分传进去。

表 1-2: openlog() 的 facility 参数

facility	说明
LOG_AUTH	安全/授权信息
LOG_AUTHPRIV	安全/授权信息（私用）
LOG_CRON	定时相关的守护进程（cron 和 at ）
LOG_DAEMON	系统守护进程
LOG_FTP	FTP 守护进程
LOG_KERN	内核信息（无法通过用户空间的进程产生）
LOG_LOCAL0 至 LOG_LOCAL7	保留由本地使用
LOG_LPT	行式打印机系统
LOG_MAIL	邮件系统
LOG_NEWS	USENET 网络新闻系统
LOG_SYSLOG	有 syslog 守护进程内部产生的消息
LOG_USER	任意的用户级消息（默认）
LOG_UUCP	UUCP 系统

1.2.2 syslog()

priority 参数可以是上面提到的 facility 与下面的 level 的组合。level 的优先级从高到低依次排序如下：

表 1-3: level 参数

level	值	说明
LOG_EMERG	0	紧急（系统不可用）（最高优先级）
LOG_ALERT	1	必须立即采取行动
LOG_CRIT	2	严重情况（如硬件设备出错）
LOG_ERR	3	出错情况
LOG_WARNING	4	警告情况
LOG_NOTICE	5	正常但重要的情况（默认值）
LOG_INFO	6	通告信息
LOG_DEBUG	7	调试信息（最低优先级）

1.3 rotate（转存/轮转）

很多时候都会让 syslog 守护进程读取到的日志信息都写入到某个文件中，随着日志的增多，文件大小会不断增大。为了避免日志文件将存储空间占满，需要限制日志文件的大小并删除过去的日志，该操作就称为 rotate（转存/轮转）。

rotate 的实现一般如下：假设 syslog 守护进程将日志写入到文件 /var/run/messages，当 messages 文件大小超过设定值时，会将 messages 中的日志信息保存到别的文件中（假设名字为 messages.0），然后清空 messages 的内容；当下一次 messages 文件的大小又超过设定值时，会再一次将 messages 中的内容保存为 messages.0，messages.0 中原有的内容则保存为 messages.1。如此类推，若干次之后就会存在 messages、messages.0、messages.1、... 、messages.n 几个文件。一般会设置 n 的最大值，超过该值的历史文件就会被删除，从而限制日志文件整体的大小。

我们可以自行编写脚本实现 rotate，可以使用专门的工具 logrotate，另外有一些 syslog 守护进程的实现自带有 rotate 的功能，如 ubox 的 logread 、busybox 的 syslogd 。

2 syslog 相关软件工具

2.1 ubox 的工具

ubox 是 OpenWrt 的工具箱，它的 syslog 系统由 `logd` 与 `logread` 两个工具实现（此处的 `logread` 是 ubox 自己的实现，与下文 busybox 提供的 `logread` 不是同一个工具）。

因为这两个工具是 OpenWrt 原生自带，它们在使用上有可能会依赖于 `procd` 和 `ubus`，目前尚未测试过在非 procd init 的环境下是否可用。

使用 procd init 时，它们通过开机脚本 `/etc/init.d/log` 自启动，具体如下：

```
#!/bin/sh /etc/rc.common
# Copyright (C) 2013 OpenWrt.org

# start after and stop before networking
START=12
STOP=89
PIDCOUNT=0

USE_PROCD=1
PROG=/sbin/logread
OOM_ADJ=-17

validate_log_section()
{
    uci_validate_section system system "${1}" \
        'log_file:string' \
        'log_size:uinteger' \
        'log_ip:ipaddr' \
        'log_remote:bool:1' \
        'log_port:port:514' \
        'log_proto:or("tcp", "udp"):udp' \
        'log_trailer_null:bool:0' \
        'log_prefix:string'
}

validate_log_daemon()
{
    uci_validate_section system system "${1}" \
        'log_size:uinteger:0' \
        'log_buffer_size:uinteger:0'
}

start_service_daemon()
{
    local log_buffer_size log_size
    validate_log_daemon "${1}"
    [ $log_buffer_size -eq 0 -a $log_size -gt 0 ] && log_buffer_size=$log_size
```

```
    [ $log_buffer_size -eq 0 ] && log_buffer_size=16
    procd_open_instance
    procd_set_param oom_adj $OOM_ADJ
    procd_set_param command "/sbin/logd"
    procd_append_param command -S "${log_buffer_size}"
    procd_set_param respawn
    procd_close_instance
}

start_service_file()
{
    PIDCOUNT="$(( ${PIDCOUNT} + 1))"
    local pid_file="/var/run/logread.${PIDCOUNT}.pid"
    local log_file log_size

    validate_log_section "${1}" || {
        echo "validation failed"
        return 1
    }
    [ -z "${log_file}" ] && return

    procd_open_instance
    procd_set_param command "$PROG" -f -F "$log_file" -p "$pid_file"
    [ -n "${log_size}" ] && procd_append_param command -S "$log_size"
    procd_close_instance
}

start_service_remote()
{
    PIDCOUNT="$(( ${PIDCOUNT} + 1))"
    local pid_file="/var/run/logread.${PIDCOUNT}.pid"
    local log_ip log_port log_proto log_prefix log_remote log_trailer_null

    validate_log_section "${1}" || {
        echo "validation failed"
        return 1
    }
    [ "${log_remote}" -ne 0 ] || return
    [ -z "${log_ip}" ] && return

    procd_open_instance
    procd_set_param command "$PROG" -f -r "$log_ip" "${log_port}" -p "$pid_file"
    case "${log_proto}" in
        "udp") procd_append_param command -u;;
        "tcp") [ "${log_trailer_null}" -eq 1 ] && procd_append_param command -0;;
    esac
    [ -z "${log_prefix}" ] || procd_append_param command -P "${log_prefix}"
    procd_close_instance
}

service_triggers()
{
    procd_add_reload_trigger "system"
    procd_add_validation validate_log_section
}

start_service()
{
    config_load system
    config_foreach start_service_daemon system
```

```
    config_foreach start_service_file system
    config_foreach start_service_remote system
}
```

可见该脚本会通过 `config_load system` 读取配置，然后将配置作为 `logd` 和 `logread` 的选项参数。具体的配置位于文件 `/etc/config/system` 中。

更为详细的说明可参考 OpenWrt 的官方文档 Runtime Logging in OpenWrt 以及 System configuration /etc/config/system 。

2.1.1 logd

`logd` 维护着一个固定大小的 ring buffer（环形缓冲区），用于保存收集到的日志（包括内核的日志）。ring buffer 的大小通过 `-S` 参数指定，可通过配置 `/etc/config/system` 中的 `log_buffer_size` 进行修改，单位为 KB。

2.1.2 logread

`logread` 用于读取 `logd` 的 ring buffer 的内容，并输出到文件或网络上的远程机器（通过 TCP/UDP 套接字）。它支持的选项有如下：

```
Usage: logread [options]
Options:
    -s <path>           Path to ubus socket
    -l  <count>         Got only the last 'count' messages
    -e  <pattern>       Filter messages with a regexp
    -r  <server> <port> Stream message to a server
    -F  <file>          Log file
    -S  <bytes>         Log size
    -p  <file>          PID file
    -h  <hostname>      Add hostname to the message
    -P  <prefix>        Prefix custom text to streamed messages
    -f                  Follow log messages
    -u                  Use UDP as the protocol
    -0                  Use \0 instead of \n as trailer when using TCP
```

- 直接执行 `logread` 会将当前 ring buffer 中的日志全打印出来（类似于 `dmesg`）。
- 加上 `-f` 则会持续地运行着，并输出 ring buffer 中新的日志。
- 使用 `-F "$log_file"` 可指定将日志输出到哪一个文件中，`-S "$log_size"` 可指定文件的大小，其中 `log_file` 和 `log_size` 都可在 `/etc/config/system` 中进行设置。（实测发现其自带有 rotate 的功能，当 `log_file` 的大小超过 `log_size` 时，会加上 “.0” 后缀转存到同一个目录下，默认只保存一份历史文件。暂未发现是否可配置保存超过一份的历史转存文件。）

2.2 busybox 的工具

busybox 自带有一些 syslog 工具，一般用到的主要为 `syslogd` 、`klogd` 和 `logread`（此处的 `logread` 与上文的 ubox 的 `logread` 不是同一个工具），均位于它 menuconfig 的 “System Logging Utilities” 下。

后文 busybox 相关的内容均基于 1.27.2 版本进行阐述。

2.2.1 syslogd

busybox 的 `syslogd` 用于读取 `/dev/log` 中的日志，并决定将其发送到文件、共享内存中的 circular buffer 或网络等位置，且其自带有简单的 rotate 功能。

它支持的特性可在 menuconfig 中进行配置，将所有特性都选上后它支持的选项如下：

```
Usage: syslogd [OPTIONS]

System logging utility

    -n              Run in foreground
    -R HOST[:PORT]  Log to HOST:PORT (default PORT:514)
    -L              Log locally and via network (default is network only if -R)
    -C[size_kb]     Log to shared mem buffer (use logread to read it)
    -K              Log to kernel printk buffer (use dmesg to read it)
    -O FILE         Log to FILE (default: /var/log/messages, stdout if -)
    -s SIZE         Max size (KB) before rotation (default 200KB, 0=off)
    -b N            N rotated logs to keep (default 1, max 99, 0=purge)
    -l N            Log only messages more urgent than prio N (1-8)
    -S              Smaller output
    -D              Drop duplicates
    -f FILE         Use FILE as config (default:/etc/syslog.conf)
```

- 特性 “Rotate message files”（FEATURE_ROTATE_LOGFILE）即为 rotate 功能，对应 `-s` 指定日志文件的限制大小以及 `-b` 指定保存多少份历史的转存文件。
- 特性 “Remote Log support”（FEATURE_REMOTE_LOG）即为网络功能的支持，对应 `-R` 和 `-L` 选项。
- 特性 “Support -D (drop dups) option”（FEATURE_SYSLOGD_DUP）对应 `-D` 选项，会丢弃掉内容相同的重复日志。判断日志是否相同不光看其主体信息，时间戳等附加的信息也会考虑在内，如 “Jan 1 08:00:00 root: foobar” 和 “Jan 1 08:00:01 root: foobar” 会被认为是两条不同的日志，只有完全相同的日志才会被丢弃掉。
- 特性 “Support syslog.conf”（FEATURE_SYSLOGD_CFG）支持使用配置文件，默认为 `/etc/syslog.conf`，也可通过 `-f` 指定其他的文件。可在配置文件中根据 `facility` 与 `level` 将日志输出到不同的目标位置，例子如下：

```
# 将所有日志输出到文件 /var/log/messages
*.* /var/log/messages
# 将所有日志输出到 console
*.* /dev/console
# 将 facility 为 LOG_KERN 的日志输出到 /var/log/kernel
kern.* /var/log/kernel
# 将 facility 为 LOG_USER 且 level 高于 LOG_NOTICE 的日志输出到 /var/log/user
user.notice /var/log/user
```

- 特性 “Read buffer size in bytes”（FEATURE_SYSLOGD_READ_BUFFER_SIZE）用于设置 `syslogd` 从 `/dev/log` 中读取内容时的 buffer 大小，它规定了单条日志消息的最大长度，超出的部分会被截断丢弃掉。
- 特性 “Circular Buffer support”（FEATURE_IPC_SYSLOG）对应 `-C[size_kb]` 选项，用于将日志送至共享内存的 circular buffer 中，可以通过 `logread` 读取出来。circular buffer 的大小可通过 “Circular buffer size in Kbytes (minimum 4KB)”（FEATURE_IPC_SYSLOG_BUFFER_SIZE）进行设置。
- 特性 “Linux kernel printk buffer support”（FEATURE_KMSG_SYSLOG）对应 `-K` 选项，用于将日志输出到 Linux 内核的 printk buffer 中，可通过 `dmesg` 读取出来。
- 剩余的一些选项：`-O` 用于指定直接将日志输出到哪个文件；`-S` 用于精简日志消息，去除 hostname、facility、level 等内容，只保留时间戳、进程名字以及消息的内容部分。

📖 说明

当前版本的 *syslogd* 中 *-f* 、*-C* 、*-O* 几个选项对应的功能是冲突的，无法同时使用。相关部分的代码如下（位于 *busybox/sysklogd/syslogd.c* 的 *timestamp_and_log* 函数中）：

```
    /* Log message locally (to file or shared mem) */
#if ENABLE_FEATURE_SYSLOGD_CFG
    {
        bool match = 0;
        logRule_t *rule;
        uint8_t facility = LOG_FAC(pri);
        uint8_t prio_bit = 1 << LOG_PRI(pri);

        for (rule = G.log_rules; rule; rule = rule->next) {
            if (rule->enabled_facility_priomap[facility] & prio_bit) {
                log_locally(now, G.printbuf, rule->file);
                match = 1;
            }
        }
        if (match)
            return;
    }
#endif
    if (LOG_PRI(pri) < G.logLevel) {
#if ENABLE_FEATURE_IPC_SYSLOG
        if ((option_mask32 & OPT_circularlog) && G.shbuf) {
            log_to_shmem(G.printbuf);
            return;
        }
#endif
        log_locally(now, G.printbuf, &G.logFile);
    }
```

可见使能了配置文件的特性后（对应宏 ENABLE_FEATURE_SYSLOGD_CFG），在读取配置文件并将日志写到目标位置后就直接 return 了，不会再执行将日志输出到共享内存区域（对应宏 ENABLE_FEATURE_IPC_SYSLOG）或直接输出到某个文件（最后的那句 log_locally(now, G.printbuf, &G.logFile)）的代码。

2.2.2 klogd

busybox 的 syslogd 无法直接获取到内核的日志信息，该功能需要通过 klogd 实现。在运行 syslogd 之后再运行 klogd 即可。

klogd 获取内核日志的方法有两种：1) 通过 klogctl() 接口；2) 通过 /proc 或设备节点。选用哪种方法可通过 menuconfig 中的 “Use the klogctl() interface”（FEATURE_KLOGD_KLOGCTL）进行设置。

klogd 在获取到内核日志后，再通过 syslog 函数将日志发送给 syslog 守护进程。

说明

虽然 *klogd* 是使用 *openlog("kernel", 0, LOG_KERN)*，但从源码中的注释来看，在 *glibc* 中 *LOG_KERN* 可能会被替换为 *LOG_USER*，因此在使用 *klogd* 过程中需要注意，内核日志的 *facility* 有可能为 *user* 而非 *kern*。

2.2.3 logread

busybox 的 logread 用于从 syslogd 共享内存的 circular buffer 中读取日志信息，它需要 syslogd 运行时带上 -C[size_kb] 选项，并且需要关闭支持配置文件的特性（FEATURE_SYSLOGD_CFG）。

2.3 syslog-ng

syslog-ng 是 syslog 守护进程的又一种实现，它本身并不依赖于 ubox 或 busybox，是一个独立的应用软件。它支持更为丰富的配置项，可以对日志进行更为灵活的处理。

syslog-ng 的配置文件为 /etc/syslog-ng.conf，详细的语法可参考官方文档 https://www.syslog-ng.com/technical-documents/list/syslog-ng-open-source-edition，下面是一份配置的例子：

```
@version:3.9

options {
    chain_hostnames(no);
    create_dirs(yes);
    flush_lines(0);
    keep_hostname(yes);
    log_fifo_size(256);
    log_msg_size(1024);
```

```
    stats_freq(0);
    flush_lines(0);
    use_fqdn(no);
    time_reopen(1); # 连接断开后等待多少秒后重新建立连接（默认为 60 秒）
    keep_timestamp(no); # 不保存日志信息自带的时间戳，用 syslog-ng 收到该日志的时间作为时间戳
};

# 定义一个 template，可使用 template 对日志的各部分内容进行处理
# 使用了此处的 template 的日志，会只显示时间戳、日志头部（程序名字等）以及主体信息，
# 相比于默认的日志信息会少了主机名
template t_without_hostname {
    template("${DATE} ${MSGHDR}${MESSAGE}\n");
};

# 定义日志的 source，即从哪里获取日志
# 此处表示从 syslog-ng 内部以及通过 Unix 数据报套接字从 /dev/log 获取日志
source src {
    internal();
    unix-dgram("/dev/log");
};

# 从 /proc/kmsg 中获取内核的日志
source kernel {
    file("/proc/kmsg" program_override("kernel"));
};

# 定义日志的 destination，即将日志送往哪里
# 此处表示将日志输出到文件 /var/log/messages，并使用刚刚定义的 template 去掉主机名
destination messages {
    file("/var/log/messages" template(t_without_hostname));
};

# 将日志输出到 console，并使用刚刚定义的 template 去掉主机名
destination console {
    file("/dev/console" template(t_without_hostname));
};

# 定义 log，用于决定将哪些 source 的日志送往哪些 destination
log {
    source(src);
    source(kernel);
    destination(messages);
    destination(console);
};
```

直接执行命令 `syslog-ng` 即可运行 syslog-ng，下面是一个 procd 式自启动脚本的例子：

```
#!/bin/sh /etc/rc.common
# Copyright (C) 2006-2016 OpenWrt.org

START=50
STOP=99

USE_PROCD=1

start_service() {
    [ -f /etc/syslog-ng.conf ] || return 1
    procd_open_instance
    procd_set_param command /usr/sbin/syslog-ng
```

```
    procd_close_instance
}

stop_service() {
    /usr/sbin/syslog-ng-ctl stop
}

reload_service() {
    stop
    start
}
```

syslog-ng 自身并不具备 rotate 的功能，无法限制日志文件的大小，一般会通过 logrotate 或自行编写脚本实现。

2.4 logrotate

logrotate 是专门用于对日志文件进行 rotate 的工具，支持将日志文件进行压缩、转存到不同目录等特性。

使用 logrotate 时需要加上配置文件的路径，如 logrotate /etc/logrotate.conf ，配置文件的语法可参考 https://jlk.fjfi.cvut.cz/arch/manpages/man/logrotate.8 。

通常可在配置文件中使用 include 命令将某个路径下所有的配置文件都包含进来，如在 /etc/logrotate.conf 中加上一句 include /etc/logrotate.d 可包含 /etc/logrotate.d 目录下的配置文件，然后该目录下可以按照不同应用、不同日志文件对配置文件进行区分，方便解耦。

以下是针对某一份日志文件进行单独配置的例子：

```
# 针对文件的 /var/log/messages 的配置，花括号中的配置项可覆盖全局配置
/var/log/messages {
    hourly          # 每小时均进行转存（实测转存周期小于一小时也可成功运行，
                    # 但如果设为 daily、weekly 等貌似在转存周期太短时会执行失败）
    size 2M         # 文件在大于 2M 时才会转存
    rotate 9        # 保存 9 份历史转存日志文件
    olddir /data    # 被转存的历史日志文件会保存到 /data 目录下
    createolddir    # 若历史日志文件的目标目录不存在则会自动创建
    compress        # 对历史日志文件进行压缩（默认使用 gzip）
    copytruncate    # 在转存时对原始日志文件复制一份后再进行截断，对复制后的文件进行转存；
                    # 而不是直接将原始日志文件移动到目标路径，避免原始日志文件的 inode 发生变化。
}
```

注意事项：

1. 配置文件的权限需要为 **0644** 或 **0444** ，否则 logrotate 执行时会有以下报错：

> error: Ignoring XXX because of bad file mode - must be 0644 or 0444.

2. logrotate 本身属于单次执行后就退出的应用程序，并非守护进程，需要借助其他守护进程（如 crond ）定期来执行。下面是 /etc/crontabs/root 的一个示例，让 root 用户每隔 3 分钟执行一次 logrotate ：

```
*/3 * * * * /usr/sbin/logrotate /etc/logrotate.conf
```

3. 一般都需要配置为 copytruncate ，除非当前使用的 syslog 守护进程支持重新打开日志文件的特性（如 busybox 的 syslogd 每秒都会重新打开日志文件），否则默认 logrotate 进行 rotate 时会直接对原始日志文件进行重命名，再创建一个与原始日志文件同名的空白文件，此时日志文件虽然名字相同但 inode 不同，而 syslog 守护进程还是继续操作原本的 inode，导致后续的日志没有正确地写入。
4. 配置为 copytruncate 时需要确保 rotate 时刻剩余的可用空间大于原始日志文件的大小。因为 copytruncate 需要先将日志文件复制一份后再进行 rotate，若剩余空间不足导致复制操作失败，后续的整个 rotate 过程也无法完成。

2.5 logger

logger 用于在 shell 中向 syslog 守护进程发送消息，使用方法类似于 echo 命令：

```
logger "foobar"
```

3 不同 syslog 方案的对比

以下针对将本地 syslog 日志写入到本地文件中的这一需求，对不同的 syslog 方案进行对比。

3.1 ubox logd + logread

优点：

- OpenWrt 原生自带，稍作配置即可使用。
- 自带获取内核日志以及简单的 rotate 功能。
- 不同于 busybox 的 `logread`，ubox 的 `logread` 可同时支持将日志写入文件和从 ring buffer 中读取日志的功能。

缺点：

- 可能依赖于 `procd` 与 `ubus`。（未测试过在缺少这两者的情况下是否可用）
- rotate 功能只支持将日志文件转存到相同目录下，且只保存一份历史文件，无压缩功能。（未发现有配置项可进行相关的设置）

3.2 busybox syslogd + klogd

优点：

- `syslogd` 自带 rotate 功能。在每次往文件写入日志之前，都会先检查文件大小是否已经超过设定的上限值，若是，则执行 rotate 操作。因为文件大小的检查是在写入日志的时候进行，而非按一定的时间间隔进行，可保证进行 rotate 时日志文件不会超出上限值很多。且因为写入日志与 rotate 是在同一进程中实现，对日志文件进行转存时直接重命名即可，不需要再复制一份，在对剩余可用空间的限制上没有 `logrotate` 的 `copytruncate` 那么大。
- `syslogd` 会保证每秒都重新打开日志文件，不需要担心文件的 inode 改变，清空日志时可随意删除日志文件，新的日志文件在下一秒就能继续正常地写入日志。

缺点：

- syslogd 本身不含获取内核日志的功能，需要额外运行 klogd 来支持。
- syslogd 不支持自定义前缀、rotate 时压缩的功能，且只能将日志文件转存到同一个目录下，无法自定义目标路径。
- 将日志写入到文件的同时无法使用 logread。

3.3 syslog-ng + logrotate

优点：

- syslog-ng 自身功能比较强大，可更为灵活地对日志进行修改、过滤，且自身带有获取内核日志的功能。
- logrotate 能实现更为灵活的 rotate 功能，如自定义目标路径、压缩日志文件等。

缺点：

- syslog-ng 本身无法监视文件大小，无法通知 logrotate 进行 rotate，只能依赖 crond 等守护进程定期地执行 logrotate ，需要权衡好日志的增长速度和定期检查的时间间隔，否则存储空间有可能会被日志占满。
- 日志文件的 inode 不能随意地被改变，否则 syslog-ng 可能无法正确地写入日志。因此：
  - logrotate 需要配置为 copytruncate ，在 rotate 时存在“复制文件”这一过程，对剩余的存储空间有一定的要求，否则 rotate 过程会失败。
  - 手动清空日志文件内容时不能直接删除日志文件，需要使用类似下面的命令：

```
echo > /var/log/messages
```

4 其他一些的注意事项

4.1 Unix 域套接字是可靠的

syslog 是靠 Unix 域套接字（Unix domain socket）实现 IPC（Inter-Process Communication，进程间通信），协议族为 AF_LOCAL（或 AF_UNIX），不管套接字的类型为字节流（SOCK_STREAM）还是数据报（SOCK_DGRAM），它都是可靠的，在使用 Unix 域套接字通信的过程中，如果读操作一端阻塞且缓冲区满了，写操作的一端也同样会阻塞，在此过程中不会有数据被丢弃。

因此，当 syslog 守护进程因为某些原因阻塞或运行耗时变长时，若此时缓冲区已经满了，有可能会影响到调用 syslog 函数的应用程序的性能。应用程序在设计时就需要考虑 syslog 函数可能的影响，不能无节制地使用 syslog 函数进行打印，也不能认为它总会很快地就执行完。

关于缓冲区，应该跟内核的套接字设置有关。对于 Unix 域数据报套接字，从测试结果来看 /proc/sys/net/unix/max_dgram_qlen 会影响其缓冲区大小，但具体的机制还不清楚。它的默认值为 10，可使用 sysctl 进行修改：

```
sysctl -w net.unix.max_dgram_qlen=XX
```

5 在 Tina 中使用 syslog

5.1 ubox 的 logd 与 logread

一般使用 procd init 的方案都会默认选上这两个工具。

logd 由 PACKAGE_logd 提供，menuconfig 中的位置为：

```
make menuconfig --->
    Base system --->
        <*> logd
```

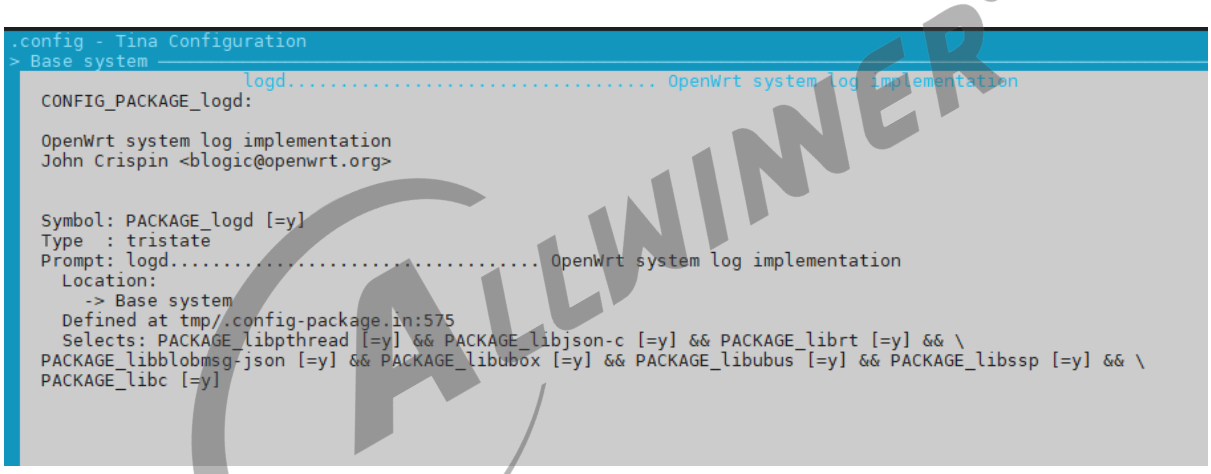

图 5-1: logd 配置图

logread 由 PACKAGE_ubox 提供，menuconfig 中的位置为：

```
make menuconfig --->
    Base system --->
        <*> ubox
```

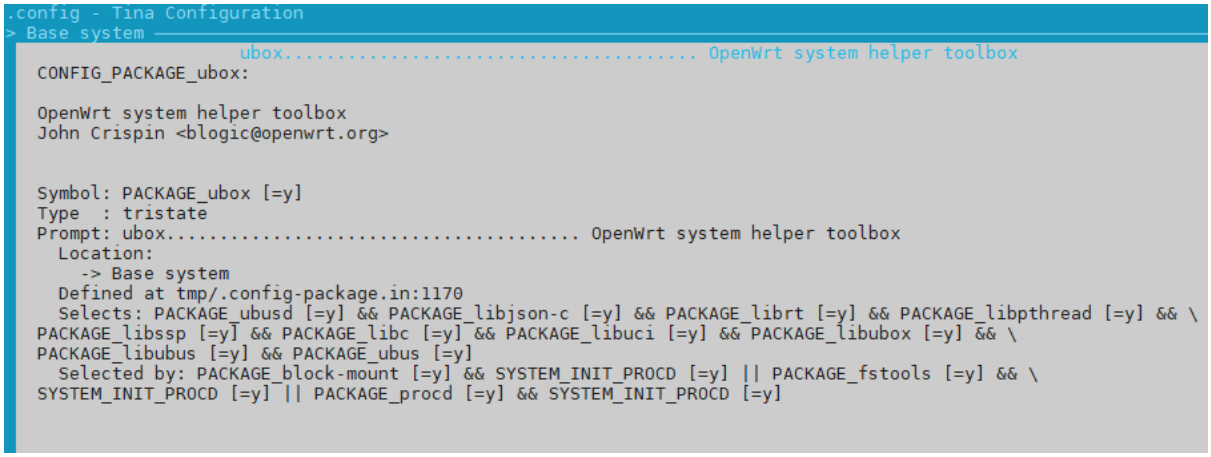

图 5-2: ubox 配置图

它们的开机脚本 `/etc/init.d/log` 由 `PACKAGE_logd` 提供；配置项位于文件 `/etc/config/system` 中，默认由 `PACKAGE_base-files` 提供，若想修改默认的配置，可以在 `target/allwinner/<方案名字>/base-files/etc/config/` 目录下放置一份自定义的 `system` 以覆盖默认的文件。

5.2 busybox 的 syslogd、klogd 与 logread

busybox 的 syslog 工具在 menuconfig 中的位置为：

```
make menuconfig --->
    Base system --->
        busybox --->
            System Logging Utilities --->
```

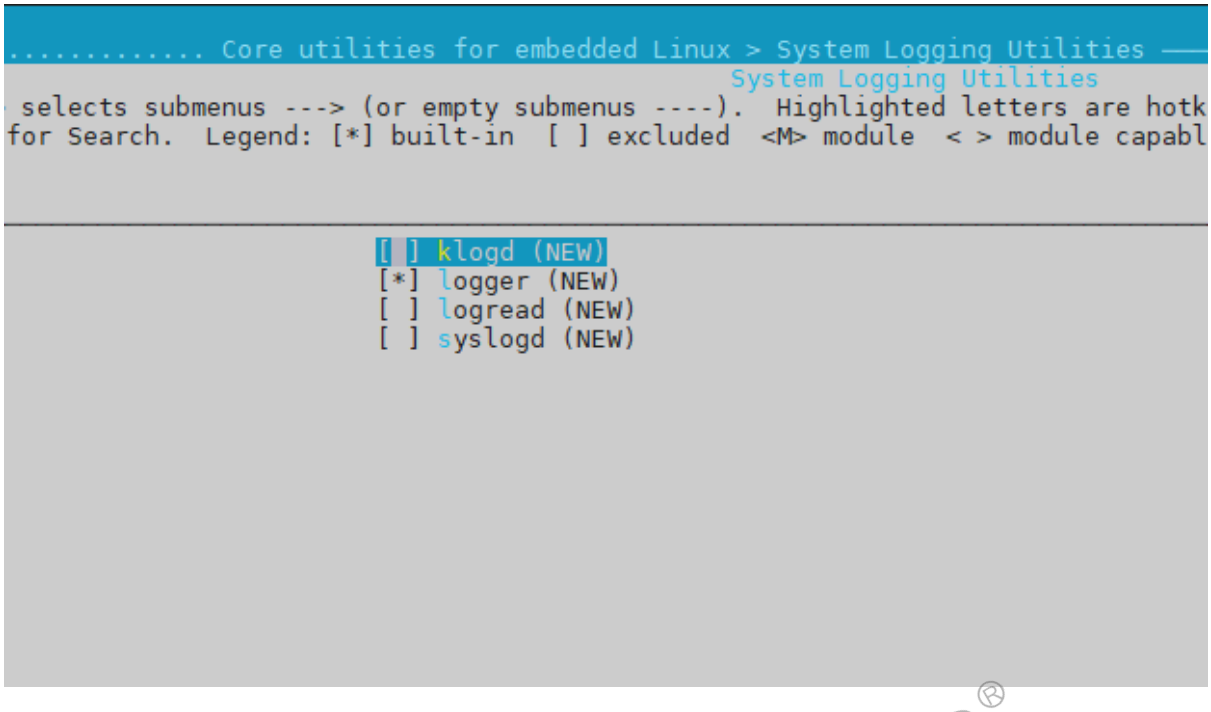

图 5-3: busybox 配置图

对应配置项的内容请参考前文的2.2 章节 。

busybox 的 syslog 工具没有自带开机脚本，若想开机自启需要自行编写。

5.3 syslog-ng

syslog-ng 在 menuconfig 中的位置为：

```
make menuconfig --->
    Administration --->
        <*> syslog-ng
```

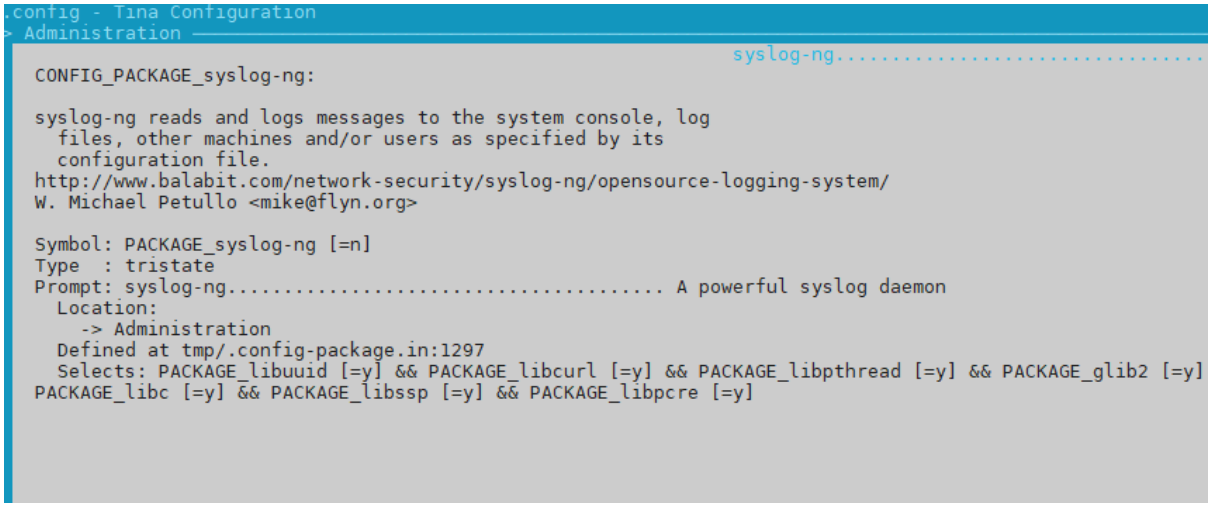

图 5-4: syslog-ng 配置图

它自带有一份 procd 式的开机脚本（会自动拷贝到小机端）以及一份配置文件的范例（不会自动拷贝到小机端），均位于 `package/admin/syslog-ng/files` 目录下。可以参考配置文件范例 `syslog-ng.conf_example` 自定义一份 `syslog-ng.conf` 放到小机端的 `/etc` 目录下。

5.4 logrotate

`logrotate` 在 menuconfig 中的位置为：

```
make menuconfig --->
    Utilities --->
        <*> logrotate
```

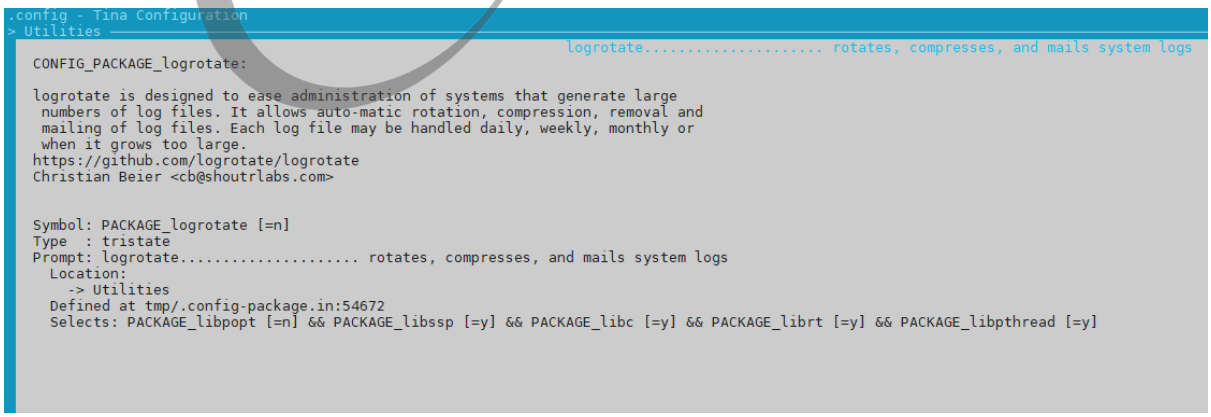

图 5-5: logrotate 配置图

它自带有一份配置文件 `logrotate.conf` ，位于 `package/utils/logrotate/files` 目录下，会自动拷贝到小机端的 `/etc` 目录下。

配置文件带有一些全局的配置项，并且会 `include /etc/logrotate.d`，因此自定义的配置可放置在小机端的 `/etc/logrotate.d` 目录下，执行 `logrotate /etc/logrotate.conf` 时会被自动调用到（注意文件的权限需要为 **0644** 或 **0444** ）。

著作权声明

商标声明

免责声明