EC2X&AG35-QuecOpen

日志系统应用向导

LTE 系列

版本： Quectel_EC2X&AG35-QuecOpen_日志系统应用向导_Preliminary

日期：2018-07-17

状态：临时文件

上海移远通信技术股份有限公司始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司
上海市徐汇区虹梅路 1801 号宏业大厦 7 楼　　邮编：200233
电话：+86 21 51086236　　邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：
http://quectel.com/cn/support/sales.htm

如需技术支持或反馈我司技术文档中的问题，可随时登陆如下网址：
http://quectel.com/cn/support/technical.htm
或发送邮件至：support@quectel.com

前言

上海移远通信技术股份有限公司提供该文档内容用以支持其客户的产品设计。客户须按照文档中提供的规范、参数来设计其产品。由于客户操作不当而造成的人身伤害或财产损失，本公司不承担任何责任。在未声明前，上海移远通信技术股份有限公司有权对该文档进行更新。

版权申明

文档历史

修订记录

版本	日期	作者	变更表述
1.0	2018-07-17	Tyler KUANG	初始版本

目录

文档历史 3
目录 4
1 引言 5
2 日志接口使用和查看 6
2.1. 日志的使用 6
2.2. 日志的查看 6
3 日志持久化配置 8
4 应用程序如何记录异常 9
4.1. 如何监听应用程序异常 9
4.2. 如何获取异常调用栈 10
5 注意事项 11

1 引言

QuecOpen 中的日志系统，采用 android 的 LOGCAT 方案。LOGCAT 包含 4 个环形读写缓冲区（MAIN，RADIO，EVENTS，SYSTEM），6 个日志等级（VERBOSE,DEBUG,INFO,WARN,ERROR,FATAL）。

在 QuecOpen 中，缓冲区的使用情况如下：

MAIN: 客户应用进程使用

RADIO: 弃用

EVENTS: 记录系统事件

SYSTEM: 记录系统关键服务进程日志

日志查看使用 logcat 命令，可根据 TAG 和日志等级进行过滤，具体用法参见 https://developer.android.com/studio/command-line/logcat

2 日志接口使用和查看

2.1. 日志的使用

日志的写入接口可以采用 android 封装的 liblog，也可以采用 QuecOpen 中封装的 qlsyslog，后续内容将介绍 qlsyslog 的使用。

头文件：qlsyslog/ql_sys_log.h

库：libql_sys_log.so

日志接口定义位于头文件：qlsyslog/ql_sys_log.h。日志接口的使用可参考下面的示例代码，并在 Makefile 中，将-lql_sys_log 添加到链接选项中。

```
#include"qlsyslog/ql_sys_log.h"

#define LOG_TAG "fortest"

int main(int argc, char *argv)
{
    QLOGV(LOG_TAG, "I am QL_SYS_LOG_VERBOSE");
    QLOGD(LOG_TAG, "I am QL_SYS_LOG_DEBUG");
    QLOGI(LOG_TAG, "I am QL_SYS_LOG_INFO");
    QLOGW(LOG_TAG, "I am QL_SYS_LOG_WARN");
    QLOGE(LOG_TAG, "I am QL_SYS_LOG_ERROR");
    QLOGF(LOG_TAG, "I am QL_SYS_LOG_FATAL");

    return 0;
}
```

2.2. 日志的查看

日志的查看使用 logcat 工具，logcat 工具的使用指南参考 https://developer.android.com/studio/command-line/logcat

下面将以 2.1 的示例代码为例，简单介绍 logcat 的使用方式。

a 通过 TAG 过滤日志：

```
/data # logcat -s fortest
--------- beginning of system
--------- beginning of main
07-17 12:24:11.472  2039  2039 V fortest : I am QL_SYS_LOG_VERBOSE
07-17 12:24:11.472  2039  2039 D fortest : I am QL_SYS_LOG_DEBUG
07-17 12:24:11.472  2039  2039 I fortest : I am QL_SYS_LOG_INFO
07-17 12:24:11.472  2039  2039 W fortest : I am QL_SYS_LOG_WARN
07-17 12:24:11.472  2039  2039 E fortest : I am QL_SYS_LOG_ERROR
07-17 12:24:11.472  2039  2039 F fortest : I am QL_SYS_LOG_FATAL
```

b 通过等级过滤日志：

```
/data # logcat -s fortest:w
--------- beginning of system
--------- beginning of main
07-17 12:24:11.472  2039  2039 W fortest : I am QL_SYS_LOG_WARN
07-17 12:24:11.472  2039  2039 E fortest : I am QL_SYS_LOG_ERROR
07-17 12:24:11.472  2039  2039 F fortest : I am QL_SYS_LOG_FATAL
```

c 通过缓冲区过滤日志

```
/data # logcat -b main
07-17 12:24:11.472  2039  2039 V fortest : I am QL_SYS_LOG_VERBOSE
07-17 12:24:11.472  2039  2039 D fortest : I am QL_SYS_LOG_DEBUG
07-17 12:24:11.472  2039  2039 I fortest : I am QL_SYS_LOG_INFO
07-17 12:24:11.472  2039  2039 W fortest : I am QL_SYS_LOG_WARN
07-17 12:24:11.472  2039  2039 E fortest : I am QL_SYS_LOG_ERROR
07-17 12:24:11.472  2039  2039 F fortest : I am QL_SYS_LOG_FATAL
```

3 日志持久化配置

服务进程 qllog 会根据过滤规则，将内核日志和应用程序日志保存到文件，配置文件位于/data/qllog.json, 配置文件为 json 格式，主要配置选项如下。

ITEM	选项	描述
log_file	必选	日志文件保存位置，注意，在系统运行中，可能会对日志文件频繁写入，不要将日志保存在关键系统分区。
rotate_file_size	必选	单个日志文件大小限制，单位 KB
rotate_file_count	必选	最大日志文件个数
log_format	可选	日志文件输出格式，可选 default，csv
kernel_priority	可选	内核日志等级，内核日志等级即 printk 的等级，可选 m(emerg), a(alert), c(crit), e(err), w(warn), n(notice), i(info), d(debug), *(所有等级)
buffer_list.{i}.name	必选	缓冲区名称，可选 main, system, events
buffer_list.{i}. filter_list.{i}.tag	可选	需要保存日志的 TAG，如果不写，默认所有 TAG
buffer_list.{i}.filter_list. {i}.priority	可选	需要保存日志的等级，可选 v(VERBOSE),d(DEBUG),i(INFO),w(WARN),e(ERROR),f(FATAL),*(所有等级)

4 应用程序如何记录异常

4.1. 如何监听应用程序异常

应用程序可以通过监听 signal 记录异常相关信息，示例代码如下。在异常处理程序中，ql_sys_log_signal(qlsyslog/ql_sys_log.h)将尽可能多的记录程序的异常信息。

```
#include <signal.h>
#include <stdlib.h>
#include"qlsyslog/ql_sys_log.h"

#define LOG_TAG "fortest"

static void handle_signal (int sig_num, siginfo_t *info, void *ptr)
{
    ql_sys_log_signal(QL_SYS_LOG_ID_MAIN, QL_SYS_LOG_FATAL, LOG_TAG, sig_num, info, ptr);
    exit(-1);
}

int main(int argc, char *argv)
{
    struct sigaction sa = {0};
    sa.sa_sigaction = handle_signal;
    sa.sa_flags = SA_SIGINFO;
    sigaction (SIGTERM, &sa, NULL);
    sigaction (SIGSEGV, &sa, NULL);
    sigaction (SIGABRT, &sa, NULL);
    sigaction (SIGINT, &sa, NULL);
    sigaction (SIGBUS, &sa, NULL);

    QLOGI(LOG_TAG, "bootup");

    /* other code */

    return 0;
}
```

4.2. 如何获取异常调用栈

应用程序发生异常时，往往希望能获取详细的应用程序调用栈，从而定位异常代码位置。ql_sys_log_signal 可以回溯函数调用栈，但同时也有一定的局限性。

方法：应用程序的编译选项中，删除-O1,-O2,-O3, -fomit-frame-pointer 优化选项，并添加-fasynchronous-unwind-tables –rdynamic 编译选项

局限：有些库在编译的时候添加了优化选项，如果异常发生在库函数，可能无法回溯详细的调用栈。

5 注意事项

在系统运行过程中，可能会产生大量的日志，会频繁写日志文件。如果日志文件存放在 flash 中，会缩短 flash 使用寿命（flash 技术参数有最大写入次数）。因此建议，在研发测试阶段，将日志保存在 flash 中，以便于调试，在后续阶段，关闭日志保存功能（不启动 qllog）或者将日志文件输出到临时文件系统。