



Tina Linux 系统调试 开发指南

版本号: 0.7
发布日期: 2022.03.12

版本历史

版本号	日期	制/修订人	内容描述
0.1	2019.01.22	AWA1225	初始版本
0.2	2019.03.25	AWA1051	添加 pstore 内容
0.3	2020.04.15	AWA1051	更新 pstore 的配置方法
0.4	2020.06.05	AWA1051	基于社区最新实现，更新 pstore 的配置方法
0.5	2021.04.17	AWA0985	完善部分章节说明
0.6	2022.01.20	AWA1450	完善 perf 章节说明
0.7	2022.03.12	AWA0985	完善 coredump 章节；添加内核日志章节



目 录

1 概述	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
2 调试方法及工具	2
2.1 内核日志	2
2.2 GDB	2
2.2.1 介绍	2
2.2.2 配置	2
2.2.3 使用	3
2.2.4 更多用法	3
2.2.5 注意事项	3
2.3 gdbserver	3
2.3.1 介绍	3
2.3.2 配置	3
2.3.3 使用	4
2.4 coredump	4
2.4.1 介绍	4
2.4.2 配置	5
2.4.3 配置生成 coredump 文件	5
2.4.4 通过 gdb 定位问题	6
2.5 perf	6
2.5.1 介绍	6
2.5.2 配置	6
2.5.3 使用	7
2.6 strace	8
2.6.1 介绍	8
2.6.2 配置	8
2.6.3 使用	8
2.7 valgrind	8
2.7.1 介绍	8
2.7.2 配置	9
2.7.3 使用	9
2.8 轻量级日志永久转存	9
2.8.1 使能日志转存	10
2.8.1.1 使能内核功能模块	10
2.8.1.2 指定分区	11
2.8.2 获取崩溃日志	12
2.8.2.1 挂载文件系统	12
2.8.2.2 读取文件	13

2.8.2.3 删除文件	13
2.8.3 高级功能配置	14
2.8.3.1 分区的空间分布	14
2.8.3.2 高级功能	14



1 概述

1.1 编写目的

本文主要服务于使用 Tina 软件平台的广大客户，帮助开发人员方便快速了解 Tina 平台系统调试工具。

1.2 适用范围

本文适用于 Tina3.5 版本以上软件平台；对硬件环境没有要求，所有 Allwinner 硬件平台都适用。

其中，注意 linux-5.4 内核上暂未支持 pstore 功能。

1.3 相关人员

适用 Tina 平台的广大客户与开发人员。

2 调试方法及工具

2.1 内核日志

内核日志默认打印在 env.cfg 中配置，文件路径：

```
文件一般在芯片方案配置目录下,例如:  
device/config/chips/v853/configs/perfl/linux/env-4.9.cfg  
device/config/chips/r528/configs/evb2/env.cfg  
  
文件中的loglevel决定打印等级  
loglevel=8
```

在进入系统后，有下面方法可以修改打印等级：

```
echo 8 > /proc/sys/kernel/printk  
dmesg -n 8
```

2.2 GDB

2.2.1 介绍

GDB(GNU symbolic debugger) 是 GNU 开源组织发布的一款调试工具，用于调试由 GCC 编译的代码。它的功能非常强大，使用命令行的调试方式，允许调试复杂的应用程序，给程序开发提供了极大的便利。

2.2.2 配置

Tina SDK 中 GDB 源码包位于 dl 目录下，默认不配置 GDB 软件包，使用时需要先选上 GDB。配置方法如下。

```
make menuconfig -->  
  Development -->  
    <*> gdb----- GNU Debugger
```

2.2.3 使用

1. 按照上述方法配置好 GDB 后，重新编译并烧写系统，在设备端口运行 gdb 即可调试应用程序。

```
gdb <process_name>
```

2.2.4 更多用法

gdb 调试命令很多，如何使用可以参考：<https://www.gnu.org/software/gdb/documentation/>

2.2.5 注意事项

- 调试信息

gdb 主要用来调试 C/C++ 的程序。在编译源码时必须要把调试信息加到可执行文件中。即编译参数带上-g 参数。如果没有-g，将看不见程序的函数名和变量名，代替它们的全是运行时的内存地址。

- 多线程调试

参考：<https://sourceware.org/gdb/onlinedocs/gdb/Forks.html>

- 已运行进程调试

gdb attach -p <pid>，其中 pid 为需要调试的进程名字。

2.3 gdbserver

2.3.1 介绍

gdbserver 是对目标设备上的程序进行远程调试的软件。

2.3.2 配置

```
make menuconfig -->
  Development -->
    <*> gdbserver..... Remote server for GNU Debugger
```

2.3.3 使用

1. 先确定本地回环接口是否打开，如未打开需要先进行网络配置，在小机端执行以下命令。

```
ip addr add dev lo 127.0.0.1/32 //设置本地回环地址为127.0.0.1  
ifconfig lo up //使能端口
```

2. 在小机端运行 gdbserver 程序

```
gdbserver 127.0.0.1:3456 process //3456为目标板端口号，用户自己定义，process为应用程序名字
```

3. 在主机端做 adb 端口映射

```
adb forward tcp:3456 tcp:3456 //第一个3456为主机端口，第二个3456为目标板端口
```

4. 在主机使用 gdb

```
${PC端编译工具链路径}/arm-openwrt-linux-gnueabi-gdb process
```

5. 主机端进行进入 gdb 界面，执行

```
target remote :3456
```

6. 连接正确可开始调试程序，最开始会从 _start 函数开始，所以可以先执行下边调试指令，进入应用程序的 main 函数进行调试。

```
b main  
c
```

2.4 coredump

2.4.1 介绍

程序运行过程中异常终止或崩溃，操作系统会将程序当时的内存状态记录下来，保存在一个文件中，这种行为就叫做 CoreDump。

可以认为 CoreDump 是内存快照，但实际上，除了内存信息之外，还有些关键的程序运行状态也会同时记录下来，例如寄存器信息（包括程序指针、栈指针等）、内存管理信息、其他处理器和操作系统状态和信息。

CoreDump 对于调试程序是非常有帮助的，因为对于有些程序错误是很难重现的，例如指针异常，而 CoreDump 文件可以再现程序出错时的情景。

💡 技巧

`man core` 可以看到 `core dump file` 详细说明。

`man 7 signal` 可以看到信号详细说明。

2.4.2 配置

tina根目录下，make kernel_menuconfig，选中以下配置。

```
Userspace binary formats -->
[*] Enable core dump support
```

涉及到的内核配置：CONFIG_COREDUMP

2.4.3 配置生成 coredump 文件

```
(1) ulimit -c unlimited;
(2) echo 'core.%e.%p' > /proc/sys/kernel/core_pattern;
```

(1) 表示在异常时产生 core dump 文件，不对 core dump 文件的大小进行限制。

(2) 指定 core dump 文件的存储位置及名称，表示产生的 core 文件中将带有崩溃的程序名、以及它的进程 ID

core_pattern的格式说明：

```
%% 单个%字符
%p 所dump进程的进程ID
%u 所dump进程的实际用户ID
%g 所dump进程的实际组ID
%s 导致本次core dump的信号
%t core dump的时间（由1970年1月1日计起的秒数）
%h 主机名
%e 程序文件名
```

具体可以通过`man core`查看

💡 技巧

`core dump` 文件默认存放在 `tmp` 目录下，如果有指定目录，注意目录必须存在，`coredump` 不支持创建目录。`/proc/sys/kernel/core_uses_pid`，内容为 1，一定会加上进程 ID，即使 `core_pattern` 中没有 `%p`。

2.4.4 通过 gdb 定位问题

生成 coredump 文件后 (例如/tmp/core), gdb 运行该文件:

```
./gdb coredump_sample /tmp/core
```

具体可以查看 gdb 或者 gdbserver 章节描述。

2.5 perf

2.5.1 介绍

Perf 是从 Linux 2.6 开始引入的一个 profiling 工具, 通过访问包括 pmu 在内的软硬件性能计数器来分析性能, 支持多架构, 是目前 Kernel 的主要性能检测手段, 和 Kernel 代码一起发布, 所以兼容性良好。

性能瓶颈如果要分类的话, 大致可以分为几个大类: **cpu/gpu/mem/storage**, 其中 gpu 用 Perf 没法直接探测 (这个目前比较好用的工具就只有 DS5), storage 一般用 tracepoint 来统计。总的说来, Perf 还是侧重于分析 cpu 的性能, 其他功能都不是很好用。常用的功能有以下几个。

- record: 收集 profile 数据
- report: 根据 profile 数据生成统计报告
- stat: 打印性能计数统计值
- top: cpu 占有率实时统计

2.5.2 配置

perf 工具依赖内核选上 PERF_EVENTS 等配置, 具体配置介绍如下:

支持 perf 基本功能选项, 必须打开:

```
CONFIG_PERF_EVENTS=y  
CONFIG_HW_PERF_EVENTS=y
```

支持堆栈跟踪:

```
CONFIG_FRAME_POINTER=y
```

支持解析内核和外部包符号, GCC 编译时加上 -g 选项:

```
CONFIG_KALLSYMS=y
```

支持 TRACEPOINTS:

```
CONFIG_FTRACE=y
```

支持内核态动态跟踪:

```
CONFIG_KPROBES=y
CONFIG_KPROBE_EVENT=y

支持用户态动态 tracepoint 跟踪：
CONFIG_DEBUG_INFO=y

//以下配置需要执行make kernel_menuconfig进行配置
支持用户态动态跟踪：
CONFIG_UPROBES=y
CONFIG_UPROBE_EVENTS=y

支持内核态 lock 跟踪：
CONFIG_LOCKDEP=y

kernel lock tracing:
CONFIG_LOCK_STAT=y

支持 TRACEPOINTS:
CONFIG_TRACEPOINTS=y
```

注意：部分内核不支持用户态动态跟踪，例如 linux3.4，具体是否支持，内核搜索是否有该配置选项即可。

上述介绍的配置都是内核的配置，Tina 中直接通过 make menuconfig 可以选上部分配置，配置方式如下：

```
tina根目录下， make menuconfig，选中以下配置：
Global build settings --->
    [*] Compile the kernel with frame pointers
    [*] Compile the kernel with symbol table information
    *- Compile the kernel with tracing support
    [*] Compile the kernel with kprobes support
Development --->
    <*> perf..... Linux performance monitoring tool
```

选上上述配置之后编译即可，部分使用者需要修改 perf 编译工具的编译参数，可配置 package/devel/perf/Makefile 中 MAKE_FLAGS 参数，修改其中的 NO_XXX=1。修改之后会新增依赖，相应的先编译依赖再编译 perf。

2.5.3 使用

```
root@TinaLinux:/# perf stat /bin/perftest
Starting convolution! thread = 4 ,count = 2
Finished convolution! Time consumed 20 seconds.
Performance counter stats for '/bin/perftest':

20236.937258 task-clock # 0.994 CPUs utilized
2404 context-switches # 0.119 K/sec
0 CPU-migrations # 0.000 K/sec
1572 page-faults # 0.078 K/sec
24241775385 cycles # 1.198 GHz
<not supported> stalled-cycles-frontend
<not supported> stalled-cycles-backend
```

```
7514299585 instructions # 0.31 insns per cycle
621110448 branches # 30.692 M/sec
1134868 branch-misses # 0.18% of all branches
20.352726051 seconds time elapsed
```

2.6 strace

2.6.1 介绍

Strace 通过 ptrace 系统调用来跟踪进程调用 syscall 的情况。

2.6.2 配置

```
tina根目录下，运行make menuconfig，选择
Utilities --->
<*> strace..... System call tracer
```

2.6.3 使用

- strace 启动程序的同时用 strace 跟踪。
- strace -p pid 对于已经启动的程序通过-p 参数 attach 上去。

2.7 valgrind

2.7.1 介绍

Valgrind 是一套 Linux 下，开放源代码 (GPLv2) 的仿真调试工具的集合。由内核 (core) 以及基于内核的其他调试工具组成。内核类似于一个框架 (framework)，它模拟了一个 CPU 环境，并提供服务给其他工具；而其他工具则类似于插件 (plug-in)，利用内核提供的服务完成各种特定的内存调试任务。Valgrind 包括以下工具，Tina 平台使用较多的工具是 memcheck，用来检查应用程序内存泄漏情况。

- Memcheck：内存使用情况检查。
- Callgrind：收集程序运行时的一些数据，函数调用关系等信息。
- Cachegrind：模拟 CPU 中的一级缓存 I1,D1 和 L2 二级缓存，能够精确地指出程序中 cache 的丢失和命中。
- Helgrind：用来检查多线程程序中出现的竞争问题。

- Massif: 堆栈分析器, 它能测量程序在堆栈中使用了多少内存, 告诉我们堆块, 堆管理块和栈的大小。

2.7.2 配置

```
tina根目录下, 运行make menuconfig, 选择  
Development -->  
    <*> valgrind .....debugging and profiling tools for linux
```

2.7.3 使用

```
valgrind --tool=memcheck --leak-check=full {program}
```

2.8 轻量级日志永久转存

全志轻量级日志永久转存方案依赖于内核原生的 pstore 文件系统, 设计了 pstore/blk 模块, 配合全志的 Flash 驱动, 实现在内核奔溃时, 自动把日志转存到 Flash 中, 并在开机后以文件形式呈现到用户空间。

此方案在全志释放的 Linux-4.9 及之后的内核版本中支持, 暂时不兼容 Linux-3.4/3.10/4.4 等旧内核版本。

pstore/blk 模块及其衍生的 **pstore/zone**, **mtdpstore** 模块已合并进 Linux 社区。详细的使用文档可参考社区内核文档。

```
Documentation/admin-guide/pstore-blk.rst
```

全志的实现支持社区的所有 Frontend 功能, 包括:

1. kmsg - 内核 Panic/Oops/emerg/restart/halt/poweroff 时的日志信息。
2. pmsg - 用户空间的信息转存 (Android 用于存储系统日志)。
3. ftrace - ftrace 信息。
4. console - 串口终端信息。

在 pstore 中, kmsg 前端基于 kmsg_dump 的机制, 在最新的版本中支持所有的 kmsg_dump_reason。kmsg_dump 机制可以在特定时机出发回调, 把内核的日志缓存 log_buf 导出。

在 pstore 中, pmsg 是 pstore 提供的用户空间转存信息的方法。用户空间程序把需要记录的信息写入到 /dev/pmsg0 的设备节点, 在重启时, 即可在 pstore 的挂载目录中获取写入的信息。在 Android 平台把 pmsg 用于存储系统日志。

当前不同存储介质对 Frontend 的支持情况如下表。

表 2-1: pstore 支持的 Frontend

介质	panic	oops	pmsg	ftrace	console
nor	N	Y	N	N	N
(ubi) spinand	N	Y	N	N	N
(nftl) spinand	Y	Y	Y	Y	Y
mmc	Y	Y	Y	Y	Y
rawnand	Y	Y	Y	Y	Y

⚠ 警告

并不是所有的 rawnand/(nftl) spinand 都支持所有的 Frontend 功能，以实际驱动为准。

2.8.1 使能日志转存

日志永久转存的方案，除了内核使能 pstore/blk 之外，还需要为其提供一个专用分区。因此使能日志转存有两个步骤。

1. 使能内核功能模块
2. 指定分区

2.8.1.1 使能内核功能模块

进入内核的 *menuconfig*，在 Tina 平台可以在任意目录执行：*m kernel_menuconfig*

```
[kernel menuconfig]
|-> File systems
    |-> Miscellaneous filesystems
        |-> [*] Persistent store support
            |-> Log panic/oops to a block device
                |-> block device identifier
                |-> Size in Kbytes of kmsg dump log to store
                |-> Maximum kmsg dump reason to store
                |-> Size in Kbytes of pmsg to store
                |-> Size in Kbytes of console to store
```

上述的属性配置，例如 **block device identifier** 可以通过 *h* 按键获取详细的说明。这些属性配置同时支持 Kconfig 和 Module Parameters 的两种配置方式，且 *Module Parameters* 具有更高的优先级。

- block device identifier
指定使用的块设备

- Size in Kbytes of kmsg dump log to store
为 kmsg 前端分配的空间大小
- Maximum kmsg dump reason to store
kmsg dumper 支持的 reason 最大值（见 enum kmsg_dump_reason）
- Size in Kbytes of pmsg to store
为 pmsg 前端分配的空间大小
- Size in Kbytes of console to store
为 console 前端分配的空间大小

💡 技巧

block device identifier 见[指定分区](#) 章节，其他属性使用默认配置即可。

2.8.1.2 指定分区

为内核 pstore/blk 模块指定使用的块设备分区，首先我们创建一个小容量分区，容量大小建议 **[256K-1M]**，参考下表。

表 2-2: pstore 分区大小建议

Flash 容量	建议大小
容量 ≤ 128M	256K
128M < 容量 ≤ 1G	512K
容量 > 1G	1M

在 sys_partition.fex 中添加 pstore 分区，例如：

```
[partition]
name       = pstore
size       = 512
user_type  = 0x8000
```

在创建了分区后，需要“告知”内核模块使用哪个分区。如上文所述，目前为止 pstore/blk 支持 Kconfig 和 Module Parameters 两种配置方式。Kconfig 比较简单，因此下文主要是讲解 Module Parameters 的配置方式。

Module Parameter 要不在手动加载模块时指定：

```
# insmod pstore_blk.ko blkdev=XXXX
```

如果是编译进内核，需要在内核 **cmdline** 中添加内核模块参数。

在全志平台，需要修改 **env-XXX.cfg**。在对应存储介质的 **setargs_XXX** 中添加如下内容。

```
pstore_blk.blkdev=<分区路径>
```

例如：

```
setargs_mmc=... pstore_blk.blkdev=/dev/mmcblk0p10 ...
```

除了路径之外，还可以使用如下的形式。

```
pstore_blk.blkdev=<主设备号:次设备号>
```

其中**主设备号**表示的存储介质，**次设备号**代指哪个分区。

我们可以在进入到命令行后，通过`ll`命令获取主次设备号，例如：

```
$ ll /dev/mmcblk0*
brw----- 1 root   root    179,  0 Jan 2 04:20 /dev/mmcblk0
brw----- 1 root   root    179, 16 Jan 2 04:20 /dev/mmcblk0boot0
brw----- 1 root   root    179, 32 Jan 2 04:20 /dev/mmcblk0boot1
brw----- 1 root   root    179,  1 Jan 2 04:20 /dev/mmcblk0p1
brw----- 1 root   root    179,  2 Jan 2 04:20 /dev/mmcblk0p2
brw----- 1 root   root    179,  3 Jan 2 04:20 /dev/mmcblk0p3
brw----- 1 root   root    179,  5 Jan 2 04:20 /dev/mmcblk0p5
```

以`/dev/mmcblk0p5`为例，**主设备号**是 **179**，**从设备号**是 **5**，因此 `cmdline` 可以写为`blkoops .blkdev=179:5`

下面进一步说明 **pstore** 分区的对应关系：

在 Tina 个别平台做了进一步封装，只需要在 `env-XXXX.cfg` 中添加`blkoops_partition=<分区名>`和`blkoops_blkdev=<分区路径|设备号>`，例如：

```
...
blkoops_partition=pstore #分区名对应sys_partition.fex
blkoops_blkdev=93:7 #可先任意写一个
setargs_nand=... pstore_blk.blkdev=${blkoops_blkdev} ...
...
```

uboot 则会根据`blkoops_partition`的分区名，自动匹配和修改`blkoops_blkdev`。

对于不支持进一步封装的方案，可在启动后查询 `cmdline` 的 `partitions` 参数，例如：

```
$ cat /proc/cmdline
.... partitions=boot-res@mmcblk0p2:env@mmcblk0p5:boot@mmcblk0p6....
```

OK，到此日志永久转存的功能已经使能。

2.8.2 获取奔溃日志

2.8.2.1 挂载文件系统

全志轻量级日志转存的方案基于的是 `pstore` 文件系统，因此需要挂载文件系统后才能使用。

在 Tina 平台中，pstore 文件系统已经实现默认开机自动挂载，可以通过 `mount` 命令确认，例如：

```
# mount
...
pstore on /sys/fs/pstore type pstore (rw,relatime)
...
```

Android 平台，需要自行实现挂载，挂载命令可参考：

```
mount -t pstore pstore /sys/fs/pstore
```

挂载后，在触发日志转存重启后，可以在挂载点 **/sys/fs/pstore** 中可获取崩溃日志文件，例如：

```
root@TinaLinux:/sys/fs/pstore# ll
drwxr-x---  2 root   root       0 Jan  1  1970 .
drwxr-xr-x  5 root   root       0 Jan  1  1970 ..
-r--r--r--  1 root   root    15504 Mar 19 19:39 dmesg-pstore-blk-0
-r--r--r--  1 root   root    15881 Mar 19 19:39 dmesg-pstore-blk-1
-r--r--r--  1 root   root       2 Jan  1  1970 pmsg-pstore-blk-0
root@TinaLinux:/sys/fs/pstore#
```

可以通过命令 `echo c > /proc/sysrq-trigger` 主动触发内核崩溃以验证功能。

2.8.2.2 读取文件

崩溃日志会以文件形式呈现到挂载点，一次崩溃一份日志，文件名格式如下。

```
<日志类型>-pstore-blk-<编号>
```

我们可通过标准的 IO 接口访问导出的日志文件。

我们可以通过名字区分 dmesg 日志记录和 pmsg 日志记录，但 dmesg 日志记录如何细分 panic/oops/oom 呢？

在 dmesg 日志记录的第一行可以进一步细分日志类型和触发次数累计，例如：

```
root@TinaLinux:/sys/fs/pstore# head -n 3 dmesg-pstore-blk-1
OOM: Total 8 times
OOM#8 Part1
<4>[ 95.111229] [<c0018e48>] (do_page_fault) from [<c0009344>] (do_PrefetchAbort+0x38/0x9c)
```

除此之外，文件时间表示的是崩溃触发时间

2.8.2.3 删除文件

可以直接删除生成的日志文件

```
rm /sys/fs/pstore/*
```

对使用 mtdpstore 模块的 spinor/(ubi) spinand 存储方案，考虑到存储物料的擦除特性，当同时存在多个连续文件，且刚好这些文件数据存储在同一个物理块内时，要把同一个块内的文件全部删除后才会真正删除文件。

2.8.3 高级功能配置

2.8.3.1 分区空间分布

默认情况下，pstore/blk 的每一份记录为 64K。意味着如果分区大小为 256K，则一共能同时存在 4 份记录。假设只使能 kmsg 和 pmsg 的记录，此时分区的划分情况大致如下表：

表 2-3: pstore 分区分布

0 - 64K	64k - 128K	128K - 192K	192K - 256 K
pmsg	dmesg.0	dmesg.1	dmesg.2

显而易见，在划分了 pmsg 的空间后，剩余的空间全部分配给 dmesg。

2.8.3.2 高级功能

内核模块通过 cmdline 中传递模块参数，可设置高级功能。日志永久转存模块支持以下模块参数。推荐使用默认配置

表 2-4: pstore 支持参数

模块名	功能	示例	默认值
pstore_blk.blkdev	供 blkloops 使用的分区	pstore_blk.blkdev=179:10	NULL
pstore_blk.oops_size	dmesg 记录大小	pstore_blk.oops_size=64	64KB
pstore_blk.pmsg_size	pmsg 记录大小	pstore_blk.pmsg_size=64	64KB
pstore_blk.console_size	console 记录大小	pstore_blk.console_size=64	64KB
pstore_blk.fttrace_size	fttrace 记录大小	pstore_blk.fttrace_size=64	64KB
pstore_blk.dump_oops	是否记录 Oops 日志	pstore_blk.dump_oops=1	True
pstore.update_ms	定时刷新日志信息	pstore.update_ms=1000	-1

💡 技巧

默认情况下，只有重启后才会刷新 pstore 的记录，除非使能了 pstore.update_ms。

著作权声明

版权所有 © 2022 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、、**全志科技**、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。