密级状态：绝密( ) 秘密( ) 内部( ) 公开(√)

Security Class: Top-Secret ( ) Secret ( ) Internal ( ) Public (√)

Rockchip Android 平台优化指导

文件状态: [ ] 草稿 [√] 正式发布 [ ] 正在修改 Status: [ ] Draft [√] Released [ ] Modifying		
	文件标识: File No.:	RK-KF-YF-290
	当前版本: Current Version：	1.0.0
	作者: Author:	卞金晨 Bian Jinchen
	完成日期: Finish Date:	2021-07-21
	审核: Auditor:	吴良清 Wu Liangqing
	审核日期: Finish Date:	2021-07-21

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。
本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

版权所有 © 2020 瑞芯微电子股份有限公司

瑞芯微电子股份有限公司
Rockchip Electronics Co., Ltd.
地址：　　福建省福州市铜盘路软件园A区18号
网址：　　www.rock-chips.com

客户服务电话： +86-4007-700-590
客户服务传真： +86-591-83951833
客户服务邮箱： fae@rock-chips.com

修订记录

版本号	作者	修改日期	修改说明	备注
V1.0.0	Bian Jinchen	2021-07-21	初始版本	

目录 Contents

1 Uboot优化

2 Kernel优化

3 Android优化

4 低内存设备优化

5 常用分析工具

1. Uboot

适当裁剪uboot中不需要的功能，可以有效提升uboot的加载速度

- 关闭loader和u-boot的串口log
- 按需求关闭avb校验的功能，以下几个config为uboot avb使能config：

```
CONFIG_ANDROID_AVB=y
CONFIG_AVB_LIBAVB=y
CONFIG_AVB_LIBAVB_AB=y
CONFIG_AVB_LIBAVB_ATX=y
CONFIG_AVB_LIBAVB_USER=y
CONFIG_RK_AVB_LIBAVB_USER=y
```

2. Kernel

开机速度优化

准备工作

如何统计内核开机耗时

从kernel第一句打印开始：

```
[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x412fd050]
```

到内核加载结束：

```
[    1.259604] Freeing unused kernel memory: 3136K
```

两个log相隔的时间为开机时间。
由于内核默认使能了`CONFIG_RK_CONSOLE_THREAD`配置，用于降低串口打印耗时对系统的影响，所以这个打印的时间并不一定100%精确，但是也能基本反映总体的耗时。

如何确认每个驱动的耗时

内核有提供initcall_debug功能用于确认每个驱动加载的耗时，可以在bootargs中添加对应的字段使能这个功能，以3568为例，添加下面的补丁即可使能对应的debug功能：

```
+++ b/arch/arm64/boot/dts/rockchip/rk3568-android.dtsi
@@ -6,7 +6,7 @@

 / {
        chosen: chosen {
-               bootargs = "earlycon=uart8250,mmio32,0xfe660000
console=ttyFIQ0";
+               bootargs = "earlycon=uart8250,mmio32,0xfe660000
console=ttyFIQ0 initcall_debug loglevel=8";
        };
```

查找耗时驱动并优化

删除某个驱动模块有两种方法：

1. 去掉这些模块对应的config选项，对于某些没有dts节点配置的模块，只能使用这种方法，比如内核的一些debug选项。
2. disable对应模块的dts节点。
   对于上面两种方法，第一种方法不仅仅减去了模块初始化的时间，并且还可以减小内核的大小，但是无法通过不同的device tree兼容不同的硬件设计，所以如果确认硬件上确实不会使用某个模块，则建议直接删掉对应的config选项，不让内核编译对应的模块。
   接下来简单描述下如何通过内核initcall的log，查找耗时大的驱动，如下面所示：

```
[    3.723183] initcall rknand_driver_init+0x0/0x40 returned -1 after 2261968
usecs
```

NAND驱动耗时2.2s，如果不需要支持NAND，可以去掉CONFIG_RK_NAND配置。

```
[    3.212658] ov5695 4-0036-1: Unexpected sensor id(000000),ret(-5)
[    3.213655] probe of 4-0036-1 returned 0 after 599822 usecs

... ...

[    3.052600] gc8034 4-0037: gc8034 read reg:0xf0 failed !
[    3.852833] gc8034 4-0037: gc8034 read reg:0xf1 failed !
[    3.852874] gc8034 4-0037: Unexpected sensor id(000000), ret(-6)
[    3.853169] probe of 4-0037 returned 0 after 1598844 usecs
```

sensor ov5695和gc8034读取id失败，如果硬件上没有接这两个sensor，可以去掉CONFIG_VIDEO_GC8034和CONFIG_VIDEO_OV5695两个配置或者dts中disable相关节点。

```
[    0.412736] calling  tracer_init_tracefs+0x0/0x1d4 @ 1
[    0.524499] initcall tracer_init_tracefs+0x0/0x1d4 returned 0 after 109063
usecs
```

这个对应的kernel配置为CONFIG_TRACING，可以禁用trace功能，该功能主要用于性能调试，会增加内核固件的大小以及内核的初始化时间，建议在正式出货版本去掉trace功能，即去掉CONFIG_FTRACE和CONFIG_TRACING这两个配置。去掉这个配置项，除了上述log中打印出的时间以外，还能够显著减小内核其他一些模块耗时。去掉这个配置，在linux4.19内核上编译会报错，请加上如下补丁：

```
diff --git a/drivers/staging/android/ion/ion_trace.h
b/drivers/staging/android/ion/ion_trace.h
index 8233691a73eb..4ac3c3786448 100644
--- a/drivers/staging/android/ion/ion_trace.h
+++ b/drivers/staging/android/ion/ion_trace.h
@@ -14,6 +14,7 @@
 #include <linux/tracepoint.h>

 #ifndef __ION_PTR_TO_HASHVAL
+#ifdef CONFIG_FTRACE
 static unsigned int __ion_ptr_to_hash(const void *ptr)
 {
        unsigned long hashval;
@@ -24,6 +25,7 @@ static unsigned int __ion_ptr_to_hash(const void *ptr)
        /* The hashed value is only 32-bit */
        return (unsigned int)hashval;
 }
+#endif
```

注意：CTS测试用户请不要修改ftrace，去除后将无法使用systrace等功能，同时产生CTS fail
内核模块的裁剪需要基于客户实际的硬件进行调整，上面只是举了一些典型的例子，主要针对一些可以禁用的模块，如果碰到一些确定需要使用的模块，并且这些模块存在开机启动耗时比较长的问题，可以有如下两个解决方案：

1. 看看能否将比较耗时的操作放在linux的work中，进行并行处理，不会阻塞驱动的初始化流程。
2. 看看能否将驱动作为ko进行加载。

性能优化

touch boost功能

该功能用于在触摸或者鼠标操作的时候cpu提速，一般默认的代码没有将速度提到最高，主要考虑到功耗和性能平衡，如果应用场景对于瞬时响应要求很高的话，可以考虑将对应频率提到最高，以rk3399 linux4.4为例:

```
diff --git a/drivers/cpufreq/cpufreq_interactive.c
b/drivers/cpufreq/cpufreq_interactive.c
index 7a627216ac39..cad6026da0f5 100644
--- a/drivers/cpufreq/cpufreq_interactive.c
+++ b/drivers/cpufreq/cpufreq_interactive.c
@@ -1329,9 +1329,9 @@ static void rockchip_cpufreq_policy_init(struct
cpufreq_policy *policy)
        if (policy->cpu == 0) {
                tunables->hispeed_freq = 1008000;
                tunables->touchboostpulse_duration_val = 500 * USEC_PER_MSEC;
-               tunables->touchboost_freq = 1200000;
```

```
+               tunables->touchboost_freq = 1416000;
        } else {
-               tunables->hispeed_freq = 816000;
+               tunables->hispeed_freq = 1800000;
        }

        index = (policy->cpu == 0) ? 0 : 1;
```

【注意】由于rk3399有大小核，所以这里两个地方都有改，如果只有一个核，请修改

```
if (policy->cpu == 0)
```

这个分支。

cpu target_loads

除了每种变频策略定义的变频方式以外，linux默认通过负载进行频率的变化，target_loads定义了变频依赖的负载大小，对应的内核节点为：

```
/sys/devices/system/cpu/cpufreq/policy0/interactive/target_loads
```

这个值越低，表明cpu会更容易变到高的频点，默认的配置是90，也就是对于每个频点，达到90%负载就往上变频。这个值的调整需要对功耗和性能进行取舍，如果客户不考虑功耗，可以将这个值降低，这样会更容易变到最高频。
但是大部分场景都需要既考虑功耗又考虑性能，所以为了在两者进行平衡，可以为多个频点配置负载，以rk3399 7.1的如下补丁为例：

```
commit 0d73bc800b4fdd9a4b5d4a31d521e1e897b107e2
Author: Wenping Zhang <wenping.zhang@rock-chips.com>
Date:   Tue Dec 25 15:21:29 2018 +0800

    change the target_loads of cpu frequence policy for cpuB.

    previous target_loads is 65, which will cause cpu increase to
    high freqence when load is low, so we change the target_loads
    to make sure cpu will transmit to high frequence when cpu load
    is high and make sure cpu will stay in low frequence when load
    is low.

    Change-Id: I3c99039ac79d5915f1186994d7c2f58ffcbdd581
    Signed-off-by: Wenping Zhang <wenping.zhang@rock-chips.com>

diff --git a/init.tablet.rc b/init.tablet.rc
index faac269..8b535ec 100755
--- a/init.tablet.rc
+++ b/init.tablet.rc
@@ -19,7 +19,7 @@ on boot
     write /dev/cpuset/background/cpus 0
     write /dev/cpuset/system-background/cpus 0-3
     write /dev/cpuset/top-app/cpus 0-5
-    write /sys/devices/system/cpu/cpufreq/policy4/interactive/target_loads 65
+    write /sys/devices/system/cpu/cpufreq/policy4/interactive/target_loads
"65 1008000:70 1200000:75 1416000:80 1608000:90"
     ioprio rt 4
```

```
 service lan_bridge_on /system/bin/sh /system/bin/lan_bridge_on.sh
```

对于target_loads，更详细的说明，请参考内核下面的文档：
Documentation/cpu-freq/governors.txt

3. Android

开机速度优化

在init.rc中，默认支持开机CPU/DDR定频，定频后开机速度有较为明显的提升，如果需要做其他操作，init.rc中添加:

```
on early-init
    do_something

on property:sys.boot_completed=1
    do_some_resume
```

优化IO效率

开机过程中，对顺序读写的数据量很大，修改预读参数能够提高IO效率，加快开机速度。SDK默认以`mmcblk2`为例，配置了几个数据读取量较大的分区。请检查自己机器的实际分区；同时，如果新增了数据读取量较大的分区，也可以参考如下做修改：

```
on late-fs
    # boot time fs tune
    write /sys/block/mmcblk2/queue/iostats 0
    write /sys/block/mmcblk2/queue/read_ahead_kb 2048
    write /sys/block/mmcblk2/queue/nr_requests 256
    write /sys/block/dm-0/queue/read_ahead_kb 2048

on property:sys.boot_completed=1
    # end boot time fs tune
    write /sys/block/mmcblk2/queue/read_ahead_kb 128
    write /sys/block/mmcblk2/queue/nr_requests 128
    write /sys/block/dm-0/queue/read_ahead_kb 128
    write /sys/block/mmcblk2/queue/iostats 1
```

- DTS可以绑定emmc的初始化顺序：

```
    aliases {
        mmc0 = &sdmmc0;
        mmc1 = &sdmmc1;
        mmc2 = &sdhci;
        mmc3 = &sdmmc2;
    };
```

检查开机过程中有无明显服务等待异常

结合bootchart及开机log，分析开机过程中的行为，若此时单线程在执行某项任务，需要去确认该任务是否影响了开机流程。

优化SELinux策略

一般为设备节点打标签，可以修改file_contexts或genfs_contexts两个中的任意一个，他们有以下特性：

1. file_contexts支持正则表达式匹配，但速度慢
2. genfs_contexts不支持正则表达式，速度快
   尽量不使用正则表达式（通配符），更多的使用genfs_contexts能够优化SELinux加载的时间。

性能优化

Pinner Service

Android 7.1及以上支持锁定特定模块到内存中，防止这些模块被反复移出/移入内存，从而提高程序的运行效率，Rockchip Android 11中默认配置了一些调用频率高的模块，位于:

```
vendor/rockchip/common/apps:
RockchipPinnerService
```

增加的这个App是overlay模块，编译时会自动区分32位系统和64位系统，想要使用此功能的产品，可以直接在`device.mk`中，添加:

```
PRODUCT_PACKAGES += RockchipPinnerService
```

1. 对于10.0及以下的版本，可以参考上述apk源码，直接修改frameworks/base中的config.xml文件，但需要自己确认32位和64位的文件。
2. 建议使用`lsof`命令查看库被占用的情况，参考代码中的库都是被system_server等进程广泛占用的，所以将其锁到内存中才会提高程序运行效率，盲目添加只会白白增加内存占用，反而导致性能下降。实际还是要根据自己的应用场景进行调整。
3. 该功能会占用一定内存，DDR容量小的设备尽量不要尝试
4. 该功能也会对安兔兔跑分有一定的提升，如果机器内存较为充足，可以考虑使能。

- 使能PinnerService后的跑分对比（RK3399挖掘机）

Android 版本	**安兔兔分数**
android 10.0	约13w
android 11.0	约14.3w

实际应用场景定向优化

具体参考AOSP说明:`https://source.android.com/devices/bootloader/boot-image-profiles`
注意: 仅仅一次的用户场景便修改profile文件，反而会使设备开机变慢，apk运行变慢。

使用32位系统

在DDR容量较小或DDR频率较低的设备中，使用32位Android对整体系统的流畅性都会有较为明显的提升。`BoardConfig.mk`中添加:

```
TARGET_ARCH := arm
TARGET_ARCH_VARIANT := armv8-2a
TARGET_CPU_ABI := armeabi-v7a
TARGET_CPU_ABI2 := armeabi

TARGET_2ND_ARCH :=
TARGET_2ND_ARCH_VARIANT :=
TARGET_2ND_CPU_ABI :=
TARGET_2ND_CPU_ABI2 :=
TARGET_2ND_CPU_VARIANT :=
```

配置系统`art`编译参数

对于产品的核心应用，可以配置一份名单来使用`引导式编译 speed-profile`，引导式编译后，App的启动速度和运行速度等都会有极大的提升。参考如下:

```
(Android 8.1+)
# 一个应用列表，用于列出哪些应用已被确定为产品的核心应用。
# 添加后将使用 speed 编译过滤器对其进行编译。
# 注意，常驻应用（如 SystemUI）只有在下次系统重新启动时才有机会使用配置文件引导型编译。
# 因此对于产品来说，让这些应用始终采用 AOT 编译可能会更好。
PRODUCT_DEXPREOPT_SPEED_APPS += \
    Settings \
    Camera2 \
    ...

# 系统服务器加载的应用的列表。这些应用将默认使用 speed 编译过滤器进行编译
PRODUCT_SYSTEM_SERVER_APPS += \
    FilesGoogle \
    Settings \
    ...
```

- 使能前后App启动时间对比（DDR 3G/630MHz）

FilesGoogle应用	am start -S -W 启动时间
PRODUCT_DEXPREOPT_SPEED_APPS未添加	约2000ms
PRODUCT_DEXPREOPT_SPEED_APPS添加后	约1100ms

注意：PRODUCT_SYSTEM_SERVER_APPS的配置，需要配合32位系统才能对低频DDR设备发挥较好的效果，否则只会增加系统占用空间，无明显效果。

- 手动编译测试方法

```
adb shell cmd package compile -m speed-profile -f com.android.settings
```

- 修改编译线程及CPU绑定
  从 Android 11 开始，Android提供了一些编译时配合CPU使用的选项，通过这些选项，编译器能够多线程地运行在特定的一组 CPU 上：

1. `dalvik.vm.boot-dex2oat-cpu-set`: 在启动时运行 dex2oat 线程的 CPU
2. `dalvik.vm.image-dex2oat-cpu-set`: 在编译启动映像时运行 dex2oat 的 CPU
3. `dalvik.vm.dex2oat-cpu-set`: 在启动后运行 dex2oat 线程的 CPU

4. `dalvik.vm.dex2oat-threads`：多线程编译
   例如：

```
PRODUCT_PRODUCT_PROPERTIES += \
    dalvik.vm.dex2oat-cpu-set=0,1,2,3 \
    dalvik.vm.boot-dex2oat-threads=4 \
    dalvik.vm.dex2oat-threads=4
```

设置编译选项时，建议设定 `dex2oat` 线程数量与选定的 CPU 数量相匹配，以避免不必要的CPU和 I/O 争用，设置多线程可以有效提高apk的安装速度，尤其是对大apk效果较明显

王者荣耀(1.9G)	time adb install *.apk
N/A	约1:44
4线程	约1:31

4. 低内存设备优化

主要针对1GB设备的内存优化。指在提升ZRAM利用率，定期回收后台进程的文件缓存和匿名页。增加优化后，系统内存整体控制在较好的水平，基本满足1GB的使用场景。
从SDK中的以下位置获取补丁包，按包中的README说明合入补丁:

```
RKDocs/android/patches/memory_optimization/MemoryReclaim_*.tar.gz
```

开机静置一分钟的可用内存(Free memory)	adb shell dumpsys meminfo
N/A	约320MB
优化后	约450M

- 补丁包中有一定的裁剪，可以按自己实际需求进行调整

5. 分析工具

bootchart

bootchart是Linux平台分析开机状态的工具，能够抓取开机过程中各个进程的启动时间及CPU/IO的使用情况，并将数据生成直观的图片，Android中也可以使用该功能对开机流程进行分析。

- 手动开启并生成图片
  可以使用bootchart抓取Android启动过程中，各个服务启动的时间、占用cpu/rom的资源情况，针对性的对Android系统进行优化，降低开机时间。

```
adb shell 'touch /data/bootchart/enabled'
adb reboot
# 开机后，pull出相关数据文件
adb shell 'tar -czf /sdcard/bootchart.tgz /data/bootchart'
adb pull /sdcard/bootchart.tgz .
bootchart bootchart.tgz -o bootchart.png
执行命令后，将会得到一幅显示各个服务占用资源情况的图。
```

- 使用SDK自带脚本
  `./system/extras/boottime_tools/bootanalyze/bootanalyze.py -h`
  使用该脚本时添加`-b`参数即可抓取bootchart.

修改CPU调度优先级和IO优先级，调整服务启动顺序，避免服务集中占用CPU/IO

init.rc中使用`priority/ioprio`关键字来调整CPU/IO的优先级

```
service xxxxx /vendor/bin/hw/xxxx
    user root
    group root system
    priority -20
    ioprio rt 0
    writepid /dev/cpuset/foreground/tasks
```

内存分析

- Android 11中，kernel支持收集dmabuf的信息，以便分析dma内存。

```
cat /d/dma_buf/bufinfo
cat /d/dma_buf/bufprocs
```

- SDK支持monkey测试，内存不足时抓取dmabuf info的新命令。可以参考以下补丁，对monkey进行修改，出问题时抓取对应的有效信息，以协助分析monkey测试时出现的问题。

```
development:
commit f82c5582bfc5f1fb9fcf46c9716e5f414bf89c84
Author: Bian Jin chen <kenjc.bian@rock-chips.com>
Date:   Fri Apr 2 16:20:38 2021 +0800

    Capture dmabuf info when the device is in low memory.

    Use: monkey ** --dump-dmabuf
    Example: monkey --throttle 200 --ignore-crashes \
            --ignore-timeouts --ignore-security-exceptions --dump-dmabuf -v -v
5000000

    Signed-off-by: Bian Jin chen <kenjc.bian@rock-chips.com>
    Change-Id: Ic0c0629158ef45db2cea4bbe30b9147cf4023363
```

使用perfetto抓取trace

Android 9.0以上，支持设备端抓取trace，pull后网页查看的方法。

1. 开发者选项中，打开System Tracing（系统跟踪）设置；也可以开启Show Quick Setting tile，在下拉状态栏中添加抓取开关。
2. 抓取后，通过adb pull出trace文件。

```
adb pull data/local/traces .
```

3. 打开网页`https://ui.perfetto.dev/#!/`
4. 选择打开trace文件，即可查看抓取的相关数据
5. 选择`Open with legacy UI`可以查看旧格式的systrace