

Unisoc Confidential For hiar

Android 10.0 设置应用客制化指导手册

文档版本

V1.0

发布日期

2020-10-15

版权所有 © 紫光展锐（上海）科技有限公司。保留一切权利。

本文件所含数据和信息都属于紫光展锐（上海）科技有限公司（以下简称紫光展锐）所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负任何与本文件相关的直接或间接的、任何伤害或损失。

请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。

Unisoc Confidential For hiar

紫光展锐（上海）科技有限公司



前言

概述

本文档主要介绍了 Android 10.0 设置模块展锐自研的功能、一些常见问题的修改方法以及如何进行自研功能的客制化等内容。

读者对象


本文档适用于所有需要在 Android 10.0 展锐平台，对设置模块进行客制化的客户和研发人员。

缩略语

缩略语	英文全名	中文解释
FW	Frameworks	Android 架构层
CTCC	China Telecom Communications Corporation	中国电信
OTA	Over-the-Air Technology	空中下载技术

符号约定

在本文中可能出现下列标志，它所代表的含义如下。

符号	说明
 说明	用于突出重要/关键信息、补充信息和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

变更信息

文档版本	发布日期	修改说明
V1.0	2020-10-15	第一次正式发布。

关键字

设置应用、智能控制、省电管理、定时开关机。

Unisoc Confidential For hiar

目 录

1 设置概览.....	1
1.1 设置主界面	1
1.2 设置代码结构	1
2 自研功能客制化介绍.....	3
2.1 闪烁指示灯功能	3
2.1.1 功能介绍	3
2.1.2 修改文件清单	3
2.1.3 客制化配置	4
2.2 智能控制功能	4
2.2.1 功能介绍	4
2.2.2 修改文件清单	6
2.2.3 客制化配置	8
2.3 运行内存显示功能	9
2.3.1 功能介绍	9
2.3.2 修改文件清单	10
2.3.3 客制化配置	10
2.4 本地系统升级功能	10
2.4.1 功能介绍	10
2.4.2 修改文件清单	11
2.4.3 客制化配置	11
2.5 CP2 版本显示功能	12
2.5.1 功能介绍	12
2.5.2 修改文件清单	12
2.5.3 客制化配置	12
2.6 定时开关机功能	13
2.6.1 功能介绍	13
2.6.2 修改文件清单	13
2.6.3 客制化配置	13
2.7 恢复出厂设置低电量提醒功能	14
2.7.1 功能介绍	14
2.7.2 修改文件清单	14
2.7.3 客制化配置	14
2.8 省电管理功能	15
2.8.1 功能介绍	15
2.8.2 修改文件清单	18
2.8.3 客制化配置	19

2.9 色温和屏幕优化功能	19
2.9.1 功能介绍	19
2.9.2 修改文件清单	20
2.9.3 客制化配置	20
2.10 护眼模式功能	21
2.10.1 功能介绍	21
2.10.2 修改文件清单	21
2.10.3 客制化配置	22
2.11 Wi-Fi MAC 获取功能	23
2.11.1 功能介绍	23
2.11.2 修改文件清单	23
2.11.3 客制化配置	24
2.12 软硬件版本号功能	24
2.12.1 功能介绍	24
2.12.2 修改文件清单	24
2.12.3 客制化配置	25
3 常见问题处理	26
3.1 时区相关问题	26
3.1.1 配置默认时区	26
3.1.2 默认时区不生效	26
3.1.3 时区更新或添加	27
3.2 时间相关问题	27
3.2.1 时间和日期设置显示错误	27
3.2.2 设置默认时间	28
3.3 语言相关问题	28
3.3.1 配置默认语言列表	28
3.3.2 配置语言支持列表	30
3.3.3 配置默认语言	31
3.4 搜索相关问题	32
3.4.1 删除一个界面的搜索	32
3.4.2 删除一个菜单的搜索	32
3.5 显示相关问题	33
3.5.1 Google 菜单的图标更换	33
3.5.2 开放源代码许可定制	34
3.5.3 流量使用情况菜单变化	35
3.6 版本号定制相关问题	37
3.6.1 软件、硬件版本号定制	37
3.6.2 内核版本号定制	37
3.7 数据库相关问题	38

3.7.1 默认字体大小修改	38
3.7.2 默认关闭系统所有动画	39
3.7.3 默认屏幕亮度修改	39
3.7.4 默认关闭快速打开相机	40
3.7.5 默认开启拿起手机即显示	41
3.7.6 默认关闭始终开启移动数据网络	41
3.7.7 配置第三方输入法为系统默认输入法	41

Unisoc Confidential For hiar

图目录

图 1-1 设置主界面	1
图 1-2 设置代码结构	2
图 2-1 闪烁指示灯开关界面	3
图 2-2 智能控制菜单及子界面	6
图 2-3 智能控制数据库字段定义	7
图 2-4 运行内存菜单	9
图 2-5 本地系统升级菜单与风险提示框	11
图 2-6 CP2 版本显示界面	12
图 2-7 定时开关机菜单及子界面	13
图 2-8 本地系统升级菜单及低电量提醒界面	14
图 2-9 省电管理功能菜单	15
图 2-10 省电优化设置功能界面	16
图 2-11 应用待机优化界面	16
图 2-12 锁屏清理应用界面	17
图 2-13 自启动管理界面	17
图 2-14 高耗电应用界面	17
图 2-15 省电管理菜单定义	18
图 2-16 省电管理子功能开关配置	19
图 2-17 色温和屏幕优化菜单与界面	20
图 2-18 护眼模式菜单与界面	21
图 2-19 护眼模式原生开关位置	22
图 2-20 SC9863A 各个 Board 配置开关位置	22
图 2-21 Wi-Fi MAC 菜单	23
图 2-22 软件版本与硬件版本菜单	24
图 3-1 系统最早时间	28
图 3-2 默认时间初始化	28

图 3-3 默认语言列表	29
图 3-4 语言支持列表	30
图 3-5 CTCC 工程引用原生语言列表路径	31
图 3-6 标记 DisplaySettings 可搜索的注解	32
图 3-7 删除一个菜单的搜索	32
图 3-8 Google 菜单	33
图 3-9 流量使用情况功能	36
图 3-10 实时网络速率显示	36
图 3-11 软件和硬件版本号配置	37
图 3-12 内核版本号获取接口	38
图 3-13 默认屏幕亮度添加 log	40
图 3-14 默认输入法配置	42
图 3-15 加载默认输入法到数据库	42

Unisoc Confidential For hiar

表目录

表 2-1 智能控制依赖关系	4
表 2-2 智能控制开关列表	8

Unisoc Confidential For hiar

1 设置概览

设置应用是 Android 系统自带的一个比较重要的应用，给用户提供了 Android 系统中一些功能的设置入口。通过对一些功能进行设置，可以让用户个性化地使用自己的设备。

这些功能既包含 Android 原生支持的功能，也包含展锐自研的功能。原生支持的功能有：网络与互联网、设备连接、应用与通知、电池管理、显示设置、存储设置、安全性设置、账号设置以及语言、输入法、日期与时间等等。展锐自研的功能有：智能控制、定时开关机、省电管理等等。

1.1 设置主界面

如图 1-1 所示，设置的主界面根据功能分类，划分了很多一级菜单项。

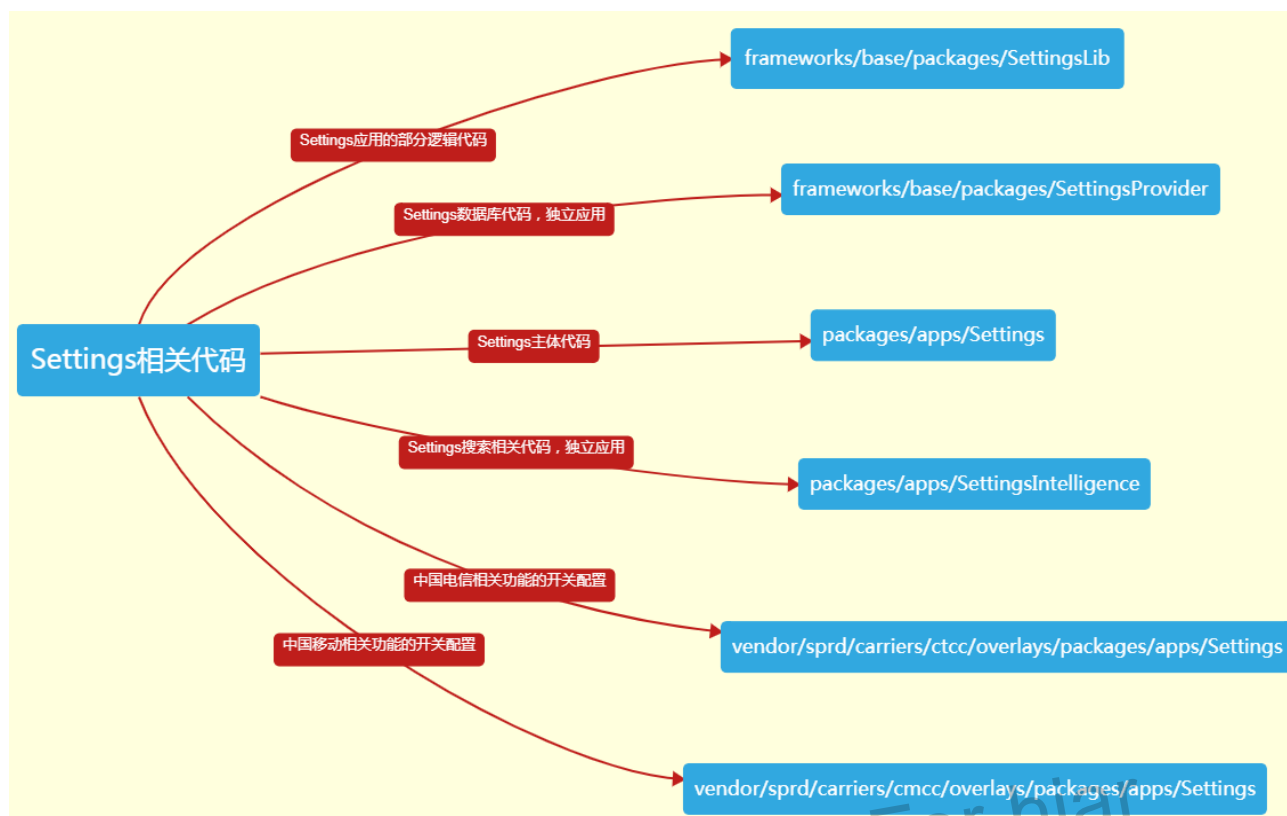
图1-1 设置主界面



1.2 设置代码结构

设置相关的代码主要包括原生功能和展锐自研功能的代码，这些代码的结构如图 1-2 所示。

图1-2 设置代码结构



2 自研功能客制化介绍

2.1 闪烁指示灯功能

2.1.1 功能介绍

Android 原生支持闪烁指示灯的功能，即设备收到通知时会闪烁通知。闪烁指示灯功能有一个功能开关 `config_intrusiveNotificationLed`，这个功能开关默认是关闭的。如果这个功能开关打开，设置应用中多处界面会显示闪烁指示灯的开关。但如果硬件上不支持指示灯功能，此时界面上显示的闪烁指示灯开关就没有任何实际功能。

故系统上设计了闪烁指示灯开关自适应功能，即在软件层面判断硬件是否支持闪烁指示灯功能，如果支持而且 `config_intrusiveNotificationLed` 开关是打开的状态，才显示闪烁指示灯开关；反之界面上不显示闪烁指示灯开关。目前设置应用中的闪烁指示灯开关界面如图 2-1 所示。

图2-1 闪烁指示灯开关界面



2.1.2 修改文件清单

- platform/frameworks/base/core/res/res/values/config.xml
将闪烁指示灯功能开关 `config_intrusiveNotificationLed` 的默认值修改为 `true`。
- platform/packages/apps/Settings/src/com/android/settings/Utils.java
添加一个公共方法 `ledFileExists()`，判断当前设备的硬件是否支持指示灯功能。

- platform/packages/apps/Settings/src/com/android/settings/notification/PulseNotificationPreferenceController.java
控制图 2-1 左边图片闪烁指示灯开关显示的控制器，在判断 config_intrusiveNotificationLed 开关值的基础上，再添加硬件是否支持指示灯功能的判断。
- platform/packages/apps/Settings/src/com/android/settings/notification/LightsPreferenceController.java
控制图 2-1 中间图片闪烁指示灯开关显示的控制器，在判断 config_intrusiveNotificationLed 开关值的基础上，再添加硬件是否支持指示灯功能的判断。
- platform/packages/apps/Settings/src/com/android/settings/notification/ZenModeVisEffectPreferenceController.java
控制图 2-1 右边图片不闪烁指示灯开关显示的控制器，在判断 config_intrusiveNotificationLed 开关值的基础上，再添加硬件是否支持指示灯功能的判断。

2.1.3 客制化配置

该功能沿用 Android 原生的配置开关 config_intrusiveNotificationLed，如果不需要该功能，可以将定义 ledFileExists() 方法的代码以及调用该方法的代码注释或者删除。

2.2 智能控制功能

2.2.1 功能介绍

该功能要求在设置应用首界面新增一个 Smart controls 菜单，用于集合各种传感器相关功能的设置开关。这些设置开关用于各个功能的打开与关闭，而相应功能的实现，依赖于各个应用的实现。它们的对应关系见表 2-1，各个功能分布在不同的界面，具体的界面显示如图 2-2 所示。

表2-1 智能控制依赖关系

功能开关			描述	功能实现模块
黑屏唤醒			黑屏时，拿起手机并双击屏幕以点亮屏幕	PhoneWindowManager
拿起手机即显示			拿起手机即可查看时间、通知和其他信息	SystemUI
智能体感	通话	体感拨号	拿起电话拨打屏幕上显示的联系人	SprdContacts
		体感接听	来电时拿起电话靠近耳朵自动接听	Telecomm
		智能切换	语音通话免提时拿起手机靠近耳朵，自动切换为听筒	Dialer
		智能通话录音	通话中，首次摇一摇手机开始录音	Dialer
		来电响铃	来电响铃时拿起手机，铃声减弱并震动	Telecomm
		来电静音	来电响铃时翻转手机转为静音	Telecomm

功能开关			描述	功能实现模块
	音视频播放	播放控制	当正在播放音乐或视频时，翻转手机播放暂停，再翻转恢复播放	NewMusic/NewGallery2
		音乐切换	正在播放音乐时，通过向左或向右摇一摇手机，切换音乐播放	NewMusic
		锁屏音乐	锁屏音乐播放时，向左或右摇一摇手机，切换音乐播放	NewMusic
	更多	智能闹钟	闹钟响铃时翻转手机转为静音	SprdDeskClock
		邮件快阅	当在电子邮件阅读时，向左或右摇手机，切换显示上一封或下一封邮件	SprdEmail
口袋模式	口袋防误触		防止手机在口袋中引起误操作	PhoneWindowManager
	铃声最大并震动		手机在口袋中来电，铃声自动最大并震动	Telecomm
	节省功耗		手机在口袋中，自动关闭 Wi-Fi 的搜索功能来节省功耗	Wi-Fi

Unisoc Confidential For hiar

图2-2 智能控制菜单及子界面



2.2.2 修改文件清单

智能控制作为一个功能开关集合，包含的内容很多，其总体实现包括 4 个部分：

- 定义各个功能开关在设置数据库中的字段名称

platform/frameworks/base/core/java/android/provider/Settings.java

所有智能控制相关的各个功能的数据库字段在 Settings.java 中是定义在一起的，具体如图 2-3 所示。

图2-3 智能控制数据库字段定义

```

/*
 * Add for Smart Controls for settings
 * @*
 */
/**{@hide}*/
public static final String SMART_WAKE = "smart_wake";
/**{@hide}*/
public static final String SMART_MOTION_ENABLED = "smart_motion_enabled";
/**{@hide}*/
public static final String EASY_DIAL = "easy_dial";
/**{@hide}*/
public static final String EASY_ANSWER = "easy_answer";
/**{@hide}*/
public static final String HANDSFREE_SWITCH = "handsfree_switch";

// 中间有部分未截取，也属于智能控制相关功能的字段定义
/**{@hide}*/
public static final String MUTE_ALARMS_SWITCH = "mute_alarms_switch";
/**{@hide}*/
public static final String SHAKE_TO_SWITCH_SWITCH = "shake_to_switch_switch";
/**{@hide}*/
public static final String QUICK_BROWSE_SWITCH = "quick_browse_switch";
/**{@hide}*/
public static final String EASY_CLEAR_MEMORY_SWITCH = "easy_clear_memory_switch";
/**{@hide}*/
public static final String TOUCH_DISABLE_SWITCH = "touch_disable_switch";
/**{@hide}*/
public static final String SMART_BELL_SWITCH = "smart_bell_switch";
/**{@hide}*/
public static final String POWER_SAVING_SWITCH = "power_saving_switch";
/*@*/

```

- 将 Android 原生支持的 3 个 Sensor 相关功能开关隐藏并移动到智能控制中
 - platform/packages/apps/Settings/src/com/android/settings/display/CameraGesturePreferenceController.java
Settings → Display → Double twist of camera（转动设备两次打开相机应用）菜单显示控制类，如果支持智能控制功能就会隐藏该菜单。
 - platform/packages/apps/Settings/src/com/android/settings/display/LiftToWakePreferenceController.java
Settings → Display → Lift to wake（拿起设备时唤醒）菜单显示控制类。如果支持智能控制功能就会隐藏该菜单，该菜单会被移动到 Settings → Smart controls → Smart wake。
 - platform/packages/apps/Settings/src/com/android/settings/gestures/PickupGesturePreferenceController.java
Settings → System → Gestures → Lift to check phone（拿起手机即显示）菜单的显示控制类。如果支持智能控制功能就会隐藏该菜单，该菜单会被移动到 Settings → Smart controls → Lift to check phone。
- 设置首界面添加菜单、添加智能控制功能开关等
 - 添加智能控制各个 Sensor 相关功能的控制开关
platform/packages/apps/Settings/res/values/config_feature_unisoc.xml
因为与 Sensor 相关的功能很多，所以对应的功能开关也很多，具体参见表 2-2:

表2-2 智能控制开关列表

功能开关	对应功能菜单	默认值
config_support_smartControls	智能控制总开关	true
config_support_smartWake	黑屏唤醒开关	true
config_support_easyDial	体感拨号开关	true
config_support_easyAnswer	体感接听开关	true
config_support_handsfreeSwitch	智能切换开关	true
config_support_smartCallRecorder	智能通话录音开关	true
config_support_easyBell	来电响铃开关	true
config_support_muteIncomingCalls	来电静音开关	true
config_support_playControl	播放控制开关	true
config_support_musicSwitch	音乐切换开关	true
config_support_lockMusicSwitch	锁屏音乐开关	true
config_support_muteAlarms	智能闹钟开关	true
config_support_quickBrowse	邮件快阅开关	true
config_support_touchDisable	口袋防误触开关	true
config_support_smartBell	铃声最大并震动开关	true
config_support_powerSaving	节省功耗开关	true

- 添加智能控制菜单功能类 SmartControlsSettings 到设置应用原生代码中
platform/packages/apps/Settings/src/com/android/settings/Settings.java
platform/packages/apps/Settings/src/com/android/settings/SettingsActivity.java
platform/packages/apps/Settings/src/com/android/settings/core/gateway/SettingsGateway.java
- 添加智能控制菜单到设置首页
platform/packages/apps/Settings/res/xml/top_level_settings.xml
- 智能控制功能实现主体
 - vendor/sprd/platform/packages/apps/Settings/src/com/sprd/settings/smartcontrols 目录下的文件。
 - vendor/sprd/platform/packages/apps/Settings/res 下还包括一些新增的资源文件等。

2.2.3 客制化配置

智能控制功能提供了统一的功能总开关和各项功能菜单的独立开关，便于客户进行定制。

- 整体去除，可以配置总的开关 config_support_smartControls 的值为 false。

- 去除其中某一项子功能，可以单独配置该项子功能开关的默认值为 `false`。

有两点情况需要注意：

- 智能控制功能的总开关打开，智能控制菜单并不一定会显示出来，它还需要如下几个 `Sensor` 至少有一个支持才可以显示：
 - `SprdSensor.TYPE_SPRDHUB_HAND_UP`
 - `SprdSensor.TYPE_SPRDHUB_SHAKE`
 - `Sensor.TYPE_PICK_UP_GESTURE`
 - `SprdSensor.TYPE_SPRDHUB_FLIP`
 - `SprdSensor.TYPE_SPRDHUB_TAP`
 - `Sensor.TYPE_WAKE_GESTURE`
 - `SprdSensor.TYPE_SPRDHUB_POCKET_MODE`
- 这些功能中一些与应用控制相关的功能开关，需要相应的展锐自研应用存在才会显示，如果是在 GMS 版本上，就可能不会显示出来，具体可以参考表 2-1 中各功能依赖的模块。

2.3 运行内存显示功能

2.3.1 功能介绍

该功能要求在设置 → 关于手机界面新增一个运行内存界面新增一个 `RAM` 菜单，用于显示当前设备的运行内存，如图 2-4 所示。

图2-4 运行内存菜单



2.3.2 修改文件清单

- platform/packages/apps/Settings/res/values/config_feature_unisoc.xml
文件中新增了该功能的控制开关：config_support_showRam。
- platform/packages/apps/Settings/res/xml/my_device_info.xml
添加该功能的菜单。
- platform/packages/apps/Settings/src/com/android/settings/deviceinfo/aboutphone/MyDeviceInfoFragment.java
将控制该功能菜单显示的控制器 SprdRamDisplayPreferenceController 实例化并添加到 Controller 列表。
- vendor/sprd/platform/packages/apps/Settings/src/com/sprd/settings/sprDRAMdisplay/Formatter.java
一个工具类，将会读取设备的运行内存并格式化后显示。
- vendor/sprd/platform/packages/apps/Settings/src/com/sprd/settings/sprDRAMdisplay/SprdRamDisplayPreferenceController.java
该功能菜单显示的控制器，根据控制开关 config_support_showRam 的值确定是否显示菜单。

2.3.3 客制化配置

该功能提供的客制化配置包括：

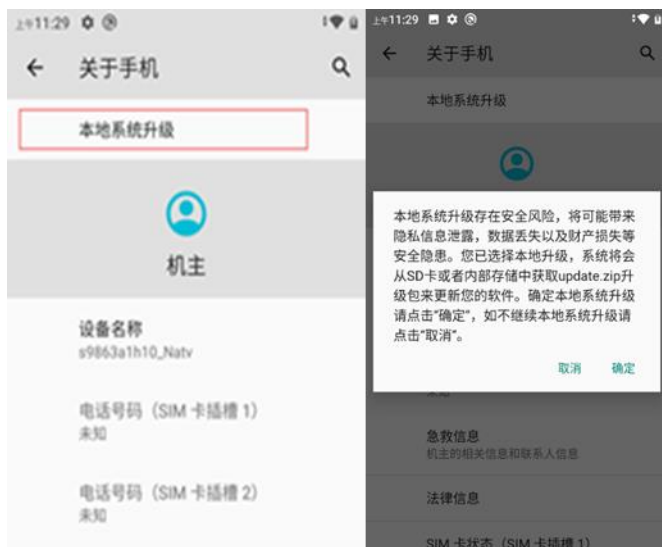
- 功能控制开关 config_support_showRam，默认值为 true。
 - 需要显示该功能，将值配置为 true。
 - 不需要该功能，将值配置为 false。
- 定制 RAM 大小显示，该功能提供了一个系统属性 ro.deviceinfo.ram，当客户为该属性配置值时，优先显示该系统属性配置的 RAM 大小。例如，将该系统属性的值配置为 1024000000，RAM 大小会显示为 1GB。

2.4 本地系统升级功能

2.4.1 功能介绍

该功能要求在设置 → 关于手机界面新增一个本地系统升级菜单，当单击该菜单进行 OTA（Over-the-Air Technology）升级时，弹出对话框提示 OTA 升级的风险，如图 2-5 所示。

图2-5 本地系统升级菜单与风险提示框



2.4.2 修改文件清单

- platform/packages/apps/Settings/res/values/config_feature_unisoc.xml
文件中新增了该功能的控制开关：config_support_otaupdate。
- platform/packages/apps/Settings/src/com/android/settings/deviceinfo/aboutphone/MyDeviceInfoFragment.java
将控制该功能菜单显示的控制器 SprdAdditionalSystemUpdatePreferenceController 实例化并添加到 Controller 列表，通过功能控制开关 config_support_otaupdate 来控制该功能菜单是否可以被搜索到。
- vendor/sprd/platform/packages/apps/Settings/src/com/android/settings/deviceinfo/SprdAdditionalSystemUpdatePreferenceController.java
该功能菜单显示的控制器，根据控制开关 config_support_otaupdate 的值确定是否动态添加功能菜单。
- vendor/sprd/platform/packages/apps/Settings/src/com/android/settings/deviceinfo/OtaUpdate.java
创建 Local software update 菜单以及点击菜单显示提醒弹出框的功能代码。

2.4.3 客制化配置

该功能提供的客制化配置就是功能控制开关 config_support_otaupdate，默认值为 true。

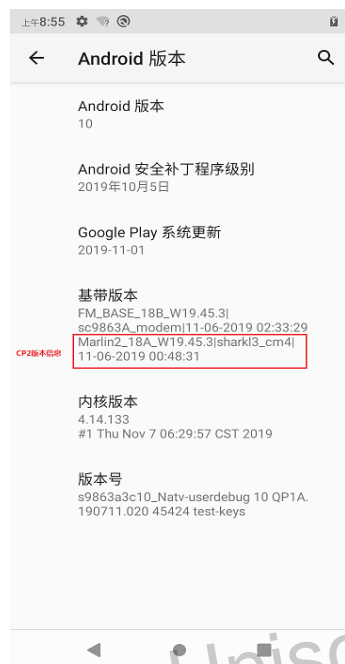
- 需要显示该功能，将值配置为 true。
- 不需要该功能，将值配置为 false。

2.5 CP2 版本显示功能

2.5.1 功能介绍

该功能要求在设置 → 关于手机 → Android 版本界面，在基带版本信息的结尾添加 CP2 的版本信息，如图 2-6 所示。

图2-6 CP2 版本显示界面



2.5.2 修改文件清单

- platform/packages/apps/Settings/res/values/config_feature_unisoc.xml
文件中新增了该功能的控制开关：config_support_showCp2Info。
- platform/packages/apps/Settings/src/com/android/settings/deviceinfo/firmwareversion/BasebandVersionDialogController.java
控制基带版本信息内容显示的控制开关，根据控制开关 config_support_showcp2info 的值确定是显示基带版本信息（系统属性 gsm.version.baseband 的值），还是显示基带版本信息与 CP2 版本信息拼接的内容。
- vendor/sprd/platform/packages/apps/Settings/src/com/android/settings/deviceinfo/SupportCPVersion.java
用于获取并解析 CP2 版本信息，并拼接到基带版本信息的功能代码。

2.5.3 客制化配置

该功能提供的客制化配置是控制开关 config_support_showCp2Info，默认值为 true。

- 要显示 CP2 版本信息，将值配置为 true。
- 不需要显示 CP2 版本信息，将值配置为 false。

2.6 定时开关机功能

2.6.1 功能介绍

该功能要求在设置应用首界面添加一个功能菜单定时开关机，实现定时开机和定时关机的功能。可以设置定时开机、关机的时间、重复周期等等，如图 2-7 所示。

图2-7 定时开关机菜单及子界面



2.6.2 修改文件清单

定时开关机功能的实现主要包括 2 个部分：

- 设置首界面添加菜单，添加功能开关
 - platform/packages/apps/Settings/res/values/config_feature_unisoc.xml
新增功能控制开关：config_support_scheduledPowerOnOff。
 - platform/packages/apps/Settings/res/AndroidManifest.xml
将定时开关机功能相关的文件添加到清单文件中。
 - platform/packages/apps/Settings/res/xml/top_level_settings.xml
设置首界面添加菜单。
- 定时开关机功能实现主体
 - vendor/sprd/platform/packages/apps/Settings/src/com/sprd/settings/timerpower 目录下的文件
定时开关机功能的实现代码。
 - vendor/sprd/platform/packages/apps/Settings/res 下新增的资源文件

2.6.3 客制化配置

该功能提供的客制化配置是控制开关 config_support_scheduledPowerOnOff，默认值为 true：

- 需要显示定时开关机功能，将值配置为 `true`。
- 不需要显示定时开关机功能，将值配置为 `false`。

2.7 恢复出厂设置低电量提醒功能

2.7.1 功能介绍

该功能要求在设置 → 系统 → 重置选项 → 清除所有数据（恢复出厂设置）界面进行恢复出厂设置操作时，如果电池电量低于 30%，需要提醒用户电量低，如图 2-8 示。

图2-8 本地系统升级菜单及低电量提醒界面



2.7.2 修改文件清单

- `platform/packages/apps/Settings/res/values/config_feature_unisoc.xml`
文件中新增了该功能的控制开关：`config_support_lowBatteryFactoryResetPrompt`。
- `platform/packages/apps/Settings/src/com/android/settings/MasterClear.java`
恢复出厂设置功能界面，在其中添加了显示提醒弹出框的代码，根据控制开关的值确定是否显示。

2.7.3 客制化配置

该功能提供的客制化配置包括两部分：

- 功能控制开关 `config_support_lowBatteryFactoryResetPrompt`，默认值为 `true`。
 - 需要显示提醒信息，将值配置为 `true`。
 - 不要显示提醒信息，将值配置为 `false`。

- 弹出提醒的电池电量阈值 BATTERY_LEVEL_LIMIT，该值在 MasterClear.java 中定义，目前默认值为 30，客户可以通过修改 BATTERY_LEVEL_LIMIT 的值变更提醒弹出的时机。

2.8 省电管理功能

2.8.1 功能介绍

该功能要求在设置 → 电池界面增加展锐自研的一系列省电管理功能。展锐自研省电管理功能主要包括：

- 省电管理功能菜单，如图 2-9 所示。

图2-9 省电管理功能菜单



- 省电优化设置功能界面，如图 2-10 所示。

图2-10 省电优化设置功能界面



- 应用待机优化界面，如图 2-11 所示。

图2-11 应用待机优化界面



- 锁屏清理应用界面，如图 2-12 所示。

图2-12 锁屏清理应用界面



- 自启动管理界面，如图 2-13 所示。

图2-13 自启动管理界面



- 高耗电应用界面，如图 2-14 所示。

图2-14 高耗电应用界面



2.8.2 修改文件清单

- platform/packages/apps/Settings/res/xml/power_usage_summary.xml
电池界面的菜单定义文件，展锐自研的省电管理功能在该文件中定义了 5 个菜单，具体如图 2-15 所示。

图2-15 省电管理菜单定义

```
<!-- UNISOC: add for power saving management start@{ -->
<Preference
    android:key="battery_saver_manage"
    android:title="@string/battery_saver_optimization"
    settings:controller="com.android.settings.fuelgauge.SprdBatterySaverManagePreferenceController"/>

<PreferenceCategory
    android:key="app_battery_saver_setting"
    android:title="@string/app_battery_saving_setting">

    <Preference
        android:key="app_battery_saver"
        android:title="@string/app_battery_saver_manager"
        settings:controller="com.android.settings.fuelgauge.SprdAppStandbyOptimizerPreferenceController"/>

    <Preference
        android:key="lock_screen_battery_save"
        android:title="@string/lock_screen_battery_save"
        settings:controller="com.android.settings.fuelgauge.SprdLockScreenBatterySaverPreferenceController"/>

    <Preference
        android:key="app_auto_run"
        android:title="@string/app_auto_run_management"
        android:fragment="com.android.settings.fuelgauge.SprdAppAutoRunFragment"
        settings:controller="com.android.settings.fuelgauge.SprdAppAutoRunManagementPreferenceController"/>

    <Preference
        android:key="power_intensive_apps"
        android:title="@string/power_intensive_apps"
        settings:controller="com.android.settings.fuelgauge.SprdPowerIntensiveAppsPreferenceController"/>
</PreferenceCategory>
<!-- UNISOC: add for power saving management end @{ -->
```

- platform/packages/apps/Settings/src/com/android/settings/fuelgauge/PowerUsageSummary.java
电池界面中添加判断，如果不支持展锐省电管理功能，会在该界面删除图 2-15 中 app_battery_saver_settings 以及其包含的 4 个菜单。
- platform/frameworks/base/packages/SettingsLib/src/com/android/settingslib/applications/ApplicationsState.java
新增一个应用过滤器 SPRD_FILTER_THIRD_PARTY。
- platform/packages/apps/Settings/src/com/android/settings/homepage/contextualcards/conditional/ConditionManager.java
增加判断条件，如果支持展锐省电管理功能，就屏蔽原生 Battery Saver 卡片的显示。
- platform/packages/apps/Settings/src/com/android/settings/fuelgauge/BatterySaverController.java
增加判断条件，如果支持展锐省电管理功能，就屏蔽原生 Battery Saver 菜单的显示。
- platform/packages/apps/Settings/res/values/config_feature_unisoc.xml
各个子功能在设置中的开关配置，具体如图 2-16 所示。

图2-16 省电管理子功能开关配置

```
<!--UNISOC: add for Power saving management
    true - support
    false - not support
    @{-->
<bool name="config_support_appStandbyOptimizer" translatable="false">true</bool>
<bool name="config_support_appAutoRunManagement" translatable="false">true</bool>
<bool name="config_support_lockScreenBatterySaver" translatable="false">true</bool>
<bool name="config_support_powerIntensiveApps" translatable="false">true</bool>
<bool name="config_support_batterySaverManage" translatable="false">true</bool>
<!-- @} -->
```

- vendor/sprd/platform/packages/apps/Settings/src/com/android/settings/fuelgauge 目录下的文件省电管理功能代码。
- vendor/sprd/platform/packages/apps/Settings/res 下对应的资源文件

2.8.3 客制化配置

该功能提供的客制化配置包括：

- 展锐自研的省电管理功能整体有一个功能总开关，总开关是由系统属性 persist.sys.pwctl.enable 控制，默认没有配置值。获取该系统属性时如果没有获取到值，默认返回 1，即该功能默认打开。如果客户不需要该功能，可以将 persist.sys.pwctl.enable 的值配置为 0，这样界面上会显示 Android 原生的省电界面。
- 在设置中为各个子功能提供了单独的配置开关，这些开关见图 2-16，如果需要支持某一子功能，就配置为 true，不需要支持就配置为 false。

2.9 色温和屏幕优化功能

2.9.1 功能介绍

该功能要求在设置 → 显示界面新增一个色温和屏幕优化菜单，提供三种颜色模式，用于调节屏幕的色域范围。其中智能环境适应模式下提供了手动调节色温的菜单，包括“冷色/自然/暖色”三个选项供用户选择，如图 2-17 所示。

图2-17 色温和屏幕优化菜单与界面



2.9.2 修改文件清单

- 设置 → 显示界面添加菜单，添加功能开关
 - platform/packages/apps/Settings/res/values/config_feature_unisoc.xml
新增功能控制开关：config_support_colorTemperatureAdjusting。
 - platform/packages/apps/Settings/res/AndroidManifest.xml
将色温和屏幕优化功能相关的文件添加到清单文件中。
 - platform/packages/apps/Settings/res/xml/display_settings.xml
显示界面添加色温和屏幕优化功能菜单。
- 颜色管理功能实现主体
 - vendor/sprd/platform/packages/apps/Settings/src/com/android/settings/display 目录下的文件
色温和屏幕优化功能的实现代码。
 - vendor/sprd/platform/packages/apps/Settings/res 下新增的资源文件

2.9.3 客制化配置

该功能提供的客制化配置有两种：

- 设置显示控制开关 config_support_colorTemperatureAdjusting，默认值为 true。
 - 需要显示色温和屏幕优化功能，配置为 true。
 - 不需要显示色温和屏幕优化功能，配置为 false。

- 颜色管理功能开关 `ro.sprd.displayenhance`，如果该系统属性配置为 `false`，即使设置应用中开关配置为 `true`，色温和屏幕优化功能也不支持。所以如果要支持色温和屏幕优化功能，两个开关需要同时配置为 `true`。

2.10 护眼模式功能

2.10.1 功能介绍

该功能要求在设置 → 显示界面将 Android 原生的夜间模式菜单修改为护眼模式菜单，用于调整屏幕色温的浓度。该功能与色温和屏幕优化功能是互斥的，当启用该功能后，色温和屏幕优化界面的功能是置灰的。如图 2-18 所示。

图2-18 护眼模式菜单与界面



2.10.2 修改文件清单

- 修改菜单的标题，从夜间模式修改为护眼模式。
- 修改夜间模式的图标。
- 互斥的功能在 FW (Frameworks) 中实现，不在设置中修改，此处不用详细说明。
- 将 Android 原生的开关 `config_nightDisplayAvailable` 配置为打开状态，原生默认为 `false`。
 - Android 原生开关位置如图 2-19 所示。

图2-19 护眼模式原生开关位置

```
xref: /sprdroid10_trunk_19c/frameworks/base/core/res/res/values/config.xml
```

Home	History	Annotate	Line#	Scopes#	Navigate#	Raw	Download
<pre><!-- Boolean indicating whether the HWC setColorTransform function can be performed efficiently in hardware. --> <bool name="config_setColorTransformAccelerated">false</bool> <!-- Boolean indicating whether the HWC setColorTransform function can be performed efficiently in hardware for individual layers. --> <bool name="config_setColorTransformAcceleratedPerLayer">false</bool> <!-- Control whether Night display is available. This should only be enabled on devices that have a HWC implementation that can apply the matrix passed to setColorTransform without impacting power, performance, and app compatibility (e.g. protected content). --> <bool name="config_nightDisplayAvailable">@bool/config_setColorTransformAccelerated</bool></pre>							

- SC9863A 各个 Board 配置开关的位置如图 2-20 所示。

图2-20 SC9863A 各个 Board 配置开关位置

/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a10c10/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	61	<bool name="config_setColorTransformAccelerated">true</bool>
/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a1h10_go/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	61	<bool name="config_setColorTransformAccelerated">true</bool>
/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a3h10/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	61	<bool name="config_setColorTransformAccelerated">true</bool>
/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a3c10_go/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	61	<bool name="config_setColorTransformAccelerated">true</bool>
/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a3h10_go/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	61	<bool name="config_setColorTransformAccelerated">true</bool>
/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a2h10/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	61	<bool name="config_setColorTransformAccelerated">true</bool>
/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a3c10/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	89	<bool name="config_setColorTransformAccelerated">true</bool>
/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a1h10/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	61	<bool name="config_setColorTransformAccelerated">true</bool>
/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a1h10_go_32b/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	61	<bool name="config_setColorTransformAccelerated">true</bool>
/sprdroid10_trunk_19c/device/sprd/sharkl3/s9863a1c10/overlay/frameworks/base/core/res/res/values/	HAD	config.xml	61	<bool name="config_setColorTransformAccelerated">true</bool>

2.10.3 客制化配置

该功能提供的客制化配置是控制开关 config_nightDisplayAvailable，默认值为 false。

- 需要该功能，配置为 true。
- 不需要该功能，配置为 false。

2.11 Wi-Fi MAC 获取功能

2.11.1 功能介绍

Wi-Fi MAC 地址显示功能是 Android 原生的功能，只是原生功能在系统未开启过 Wi-Fi 的情况下，无法获取到 MAC 地址，只能显示未知。而 CTCC（China Telecom Communications Corporation）入库要求在未开启过 Wi-Fi 开关的情况下必须能获取到 MAC 地址，因此提出了该功能的需求。该功能在设置部分的实现主要包括如下几点：

- 设计一个开关，只在 CTCC 版本上才生效，其他版本沿用 Android 原生功能。
- 提供在未开启 Wi-Fi 开关的情况下获取 Wi-Fi MAC 地址的功能。

Wi-Fi MAC 菜单如图 2-21 所示。

图2-21 Wi-Fi MAC 菜单



2.11.2 修改文件清单

- platform/frameworks/base/packages/SettingsLib/res/values/carrier_configs.xml
文件中新增了该功能的控制开关：config_enableGetWifiMacFromFile，默认为 false。
- platform/frameworks/base/packages/SettingsLib/src/com/android/settingslib/deviceinfo/AbstractWifiMacAddressPreferenceController.java
添加了获取 Wi-Fi MAC 地址的逻辑。
- vendor/sprd/carriers/ctcc/overlays/packages/apps/Settings/路径所有文件
动态 overlay 用于在 CTCC 版本将控制开关 config_enableGetWifiMacFromFile 的值动态 overlay 为 true。
- vendor/sprd/feature_configs/carriers/ctcc/config.mk
配置是否编译动态 overlay 的 apk。

2.11.3 客制化配置

该功能仅在 CTCC 版本生效，所以编译其他版本不包含此功能。如果在 CTCC 版本上需要去除该功能，只需要在 `vendor/sprd/carriers/ctcc/overlays/packages/apps/Settings/res/carrier_configs.xml` 中将控制开关配置为 `false` 即可。

2.12 软硬件版本号功能

2.12.1 功能介绍

在设备上显示软件版本号和硬件版本号是 CTCC 入库的要求，该功能在设置部分的实现主要包括如下几点：

- 在设置 → 关于手机界面新增一个软件版本菜单，用于显示软件版本号。
- 在设置 → 关于手机界面新增一个硬件版本菜单，用于显示硬件版本号。
- 设计一个开关，只在 CTCC 版本上才生效，其他版本不显示软件版本菜单和硬件版本菜单。
- 在 CTCC 版本上提供软件版本菜单和硬件版本菜单的搜索功能。

具体实现效果如图 2-22 所示。

图2-22 软件版本与硬件版本菜单



2.12.2 修改文件清单

- `platform/packages/apps/Settings/res/values/carrier_configs.xml`
文件中新增了该功能的控制开关：`config_enableCtccVersion`，默认为 `false`。
- `platform/packages/apps/Settings/res/xml/my_device_info.xml`

添加软件版本号和硬件版本号菜单。

- vendor/sprd/platform/packages/apps/Settings/src/com/android/settings/deviceinfo/HardwareRevisionPreferenceController.java
硬件版本号控制器。
- vendor/sprd/platform/packages/apps/Settings/src/com/android/settings/deviceinfo/SoftwareRevisionPreferenceController.java
软件版本号控制器。
- vendor/sprd/feature_configs/carriers/ctcc/config.mk
配置是否编译动态 overlay 的 apk。
- vendor/sprd/carriers/ctcc/overlays/packages/apps/Settings/路径所有文件
动态 overlay 用于在 ctcc 版本将控制开关 config_enableCtccVersion 的值动态 overlay 为 true。

2.12.3 客制化配置

该功能仅在 CTCC 版本生效，所以编译其他版本不包含此功能。如果在 CTCC 版本上需要去除该功能，需要在 vendor/sprd/carriers/ctcc/overlays/packages/apps/Settings/res/carrier_configs.xml 中将控制开关配置为 false 即可。

Unisoc Confidential For hiar

3 常见问题处理

3.1 时区相关问题

3.1.1 配置默认时区

设置默认时区，需要配置系统属性 `persist.sys.timezone` 的值，该系统属性需要在 `device/sprd` 目录下，产品对应 board 的 `system.prop` 文件中添加，然后编译版本即可。

3.1.2 默认时区不生效

配置的默认时区不生效，通常有如下几种情况：

1. 自动更新时区默认开启，插卡开机，有网络连接，开机后自动更新了时区，导致默认时区不生效。

情况说明：自动更新时区默认开启情况下，通过网络更新时区是正常的。如果要想让其不更新时区，可以默认关闭自动更新时区开关。

修改方法：将自动更新时区开关 `def_auto_time_zone` 的值设置为 `false`。

文件路径：`platform/frameworks/base/packages/SettingsProvider/res/values/defaults.xml`

2. 自动更新时区默认开启，开机不插卡不连接网络，开机后自动更新了时区，导致默认时区不生效。

情况说明：自动更新时区默认开启情况下，不插卡开机，不连接网络，设备也会驻留到紧急网络上，此时会根据紧急网络上报的国家码进行匹配并更新时区。

修改方法：

- 可以参考第 1 种情况的修改方法关闭自动更新时区开关。
- 通常客户会希望保持自动更新时区开关打开，在不插 SIM 卡的情况下开机不更新时区，如果要想实现这种功能，需要如下修改：

在 `frameworks/opt/telephony/src/java/com/android/internal/telephony/NewNitzStateMachine.java` 文件的 `setAndBroadcastNetworkSetTimeZone` 方法开头添加如下代码，对应的类也需要导入：

```
+ import com.android.internal.telephony.uicc.IccCardApplicationStatus.AppState;
+ import com.android.internal.telephony.uicc.UiccCardApplication;

+     UiccCardApplication uiccApp = mPhone.getUiccCardApplication();
+     if (uiccApp == null || uiccApp.getState() == AppState.APPSTATE_UNKNOWN) {
+         Rlog.d(LOG_TAG, "Not to set network time zone due to sim absent.");
+         return;
+     }
```

3. 设置的默认时区不在支持的时区列表中，显示了其他时区，导致默认时区不生效。

情况说明：系统支持的默认时区列表在 tzlookup.xml 文件中，如果是 tzlookup.xml 中没有的时区 id，配置默认时区是无效的，因为系统不支持。

修改方法：配置系统支持的时区。

文件路径：system/timezone/output_data/android/tzlookup.xml

4. 系统带有 Google 开机向导。Google 开机向导中有时区设置界面，如果设置的默认时区不在其列表中，开机向导会显示其他时区，这时如果点击下一步，会将开机向导界面显示的时区设置到系统中，导致默认时区不生效。

情况说明：Google 开机向导应用维护的时区列表是独立的，与系统支持的时区列表可能不一致。

修改方法：Google 开机向导应用是闭源的，所以无法修改，可以配置 Google 开机向导应用支持的时区。

5. SELinux 权限异常，导致默认时区不生效。

情况说明：因为是权限问题，所以需要添加相应的权限。

修改方法：在 device/sprd/XXX/common/sepolicy/vendor_init.te 文件中，添加权限方法如下：

```
allow vendor_init exported_system_prop:property_service { set }
```

说明

如果是因为 SELinux 权限问题不生效，kernel log 中会有如下信息打印，可以作为参考：

```
selinux: avc: denied { set } for property=persist.sys.timezone pid=1 uid=0 gid=0 scontext=u:r:vendor_init:s0
tcontext=u:object_r:exported_system_prop:s0 tclass=property_service permissive=0
.....init: Unable to set property 'persist.sys.timezone' to 'Europe/Amsterdam' in property file '/vendor/build.prop': SELinux permission check failed
```

3.1.3 时区更新或添加

Android 10.0 时区数据更新已经 mainline，所以无法修改。external/icu 仓库下如果修改，可以通过执行脚本 external/icu/tools/updateicudata.py 生成新的 icudt63l.dat 文件，然后全编译即可生效。

3.2 时间相关问题

3.2.1 时间和日期设置显示错误

问题：设置中关闭使用网络提供的时间开关后，设置日期为 2007 年 1 月 1 日，确定后显示的日期是 2007 年 11 月 5 日，时间为上午 8 点。

原因：这是由于 Android 原生设置的代码中限制了可以设置的最早时间是 2007 年 11 月 5 日 0 点，因为中国的北京时间是东八区 GMT+8，所以显示的时间是上午 8 点，如图 3-1 所示。

图3-1 系统最早时间

/sprdroidq_trunk/packages/apps/Settings/src/com/android/settings/datetime/UpdateTimeAndDateCallback.java

ie | History | Annotate | Line# | Scopes# | Navigate# | Raw | Download

```

/*
 * Copyright (C) 2016 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.android.settings.datetime;

import android.content.Context;

public interface UpdateTimeAndDateCallback {
    // Minimum time is Nov 5, 2007, 0:00.
    long MIN_DATE = 1194220800000L;

    void updateTimeAndDateDisplay(Context context);
}
    
```

3.2.2 设置默认时间

Android 10.0 在 AlarmManagerService 中做了限制，设备的系统时间不能早于设备烧录版本的生成时间，如果要设置更早的时间作为默认时间将无法生效，会被 AlarmManagerService 强制更改为版本生成时间。具体代码在 AlarmManagerService 的 onStart 方法中，如图 3-2 所示。

图3-2 默认时间初始化

```

// Ensure that we're booting with a halfway sensible current time. Use the
// most recent of Build.TIME, the root file system's timestamp, and the
// value of the ro.build.date.utc system property (which is in seconds).
final long systemBuildTime = Long.max(
    1000L * SystemProperties.getLong("ro.build.date.utc", -1L),
    Long.max(Environment.getRootDirectory().lastModified(), Build.TIME));
if (mInjector.getCurrentTimeMillis() < systemBuildTime) {
    Slog.i(TAG, "Current time only " + mInjector.getCurrentTimeMillis()
        + ", advancing to build time " + systemBuildTime);
    mInjector.setKernelTime(systemBuildTime);
}
    
```

3.3 语言相关问题

3.3.1 配置默认语言列表

默认语言列表是指设置语言界面可以直接移动并进行设置的语言列表，不需要手动添加，如图 3-3 所示。

图3-3 默认语言列表



如果要配置默认语言列表，需要配置设置数据库中 Settings.System.SYSTEM_LOCALES 配置项的默认值。具体修改可以分为两种情况：

非 GMS 版本，没有 Google 开机向导

非 GMS 版本可以通过直接修改 SettingsProvider 配置默认语言列表。具体修改步骤为：

步骤 1 在 SettingsProvider/res/values/defaults.xml 中定义默认值字符串，如下所示。其中第一个语言会作为默认语言被选中，它会覆盖掉 PRODUCT_LOCALES 中的配置的默认语言。

```
<string name="system_locales" translatable="false"> fr-FR,en-US,es-ES,zh-Hans-CN</string>
```

步骤 2 将上面配置的默认值字符串加载到设置数据库中，配置项在 System 表中，所以需要在文件路径 SettingsProvider/src/com/android/providers/settings/DatabaseHelper.java 的 loadSystemSettings 方法中添加如下代码：

```
loadStringSetting(stmt, Settings.System.SYSTEM_LOCALES, R.string.system_locales);
```

---结束

GMS 版本，有 Google 开机向导

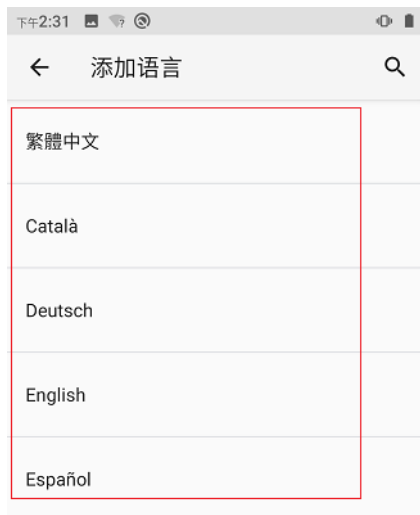
如果有 Google 开机向导，按照第一种情况修改，则 Google 开机向导的语言设置界面默认选中的是配置的第一个语言，不论是直接使用这个语言，还是选择其他语言，进入下一步设置到系统后，SYSTEM_LOCALES 的值会被覆盖为设置的语言，默认语言列表里也只有设置的语言一个。

这种情况下如果要设置默认语言列表，可以考虑在开机向导结束后，在 SystemUI 中再次写入期望设置的语言列表。但是由于 Google 开机向导中用户在语言设置界面会选择并设置自己的语言，如果此时再配置默认语言列表，且语言列表中第一个语言与用户设置的语言不一样，会将用户设置的语言给覆盖，因此不建议如此修改。

3.3.2 配置语言支持列表

语言支持列表是指添加语言界面可以添加的语言列表，如图 3-4 示。

图3-4 语言支持列表



支持语言列表是由宏 `PRODUCT_LOCALES` 配置的，每个工程配置都可能不同，如何查找一个工程对应的 `PRODUCT_LOCALES` 配置，下面以 UMS512 平台的 ums512 1h10 oversea 工程作为示例，步骤为：

步骤 1 在 `device/sprd/sharkl5pro/ums512_1h10/` 目录下找到 ums512 1h10 Native 工程对应的 mk 文件 `ums512_1h10_Natv.mk`。在 mk 文件中搜索 **PRODUCT_REVISION**，查看其配置的内容是什么，该工程配置的值是 **oversea multi-lang**，这两个值都是指向 `vendor/sprd/feature_configs/` 下的目录或子目录，例如：

- **oversea** 在 `vendor/sprd/feature_configs/carriers/` 目录
- **multi-lang** 在 `vendor/sprd/feature_configs/` 目录

步骤 2 查看指向的路径下相应的 config.mk 文件，其中 multi-lang 的 config.mk 配置如下两个宏的值：

- **FEATURES.PRODUCT_LOCALES**：这个宏是对 `PRODUCT_LOCALES` 的覆盖，其配置的值就是语言支持列表。
- **FEATURES.PRODUCT_PACKAGE_OVERLAYS**：这个宏配置的是 overlay 目录，如果其 overlay 目录下存在 `frameworks/base/core/res/res/values/locale_config.xml` 文件，则该文件配置的是当前工程加载的语言资源列表。如果语言支持列表中的某个语言在该文件中被注释了，则在设置中切换这个语言时语言不会变化，因为该语言对应的资源在系统中没有加载。
- 查看 oversea 的 config.mk 文件，没有与语言相关的配置。

---结束

上面的示例是 ums512 1h10 oversea 工程，有些工程没有配置 overlay 目录，或者没有配置 `FEATURES.PRODUCT_PACKAGE_OVERLAYS` 属性，这种情况下当前工程加载的语言资源列表就是 Android 原生的 `frameworks/base/core/res/res/values/locale_config.xml` 文件。

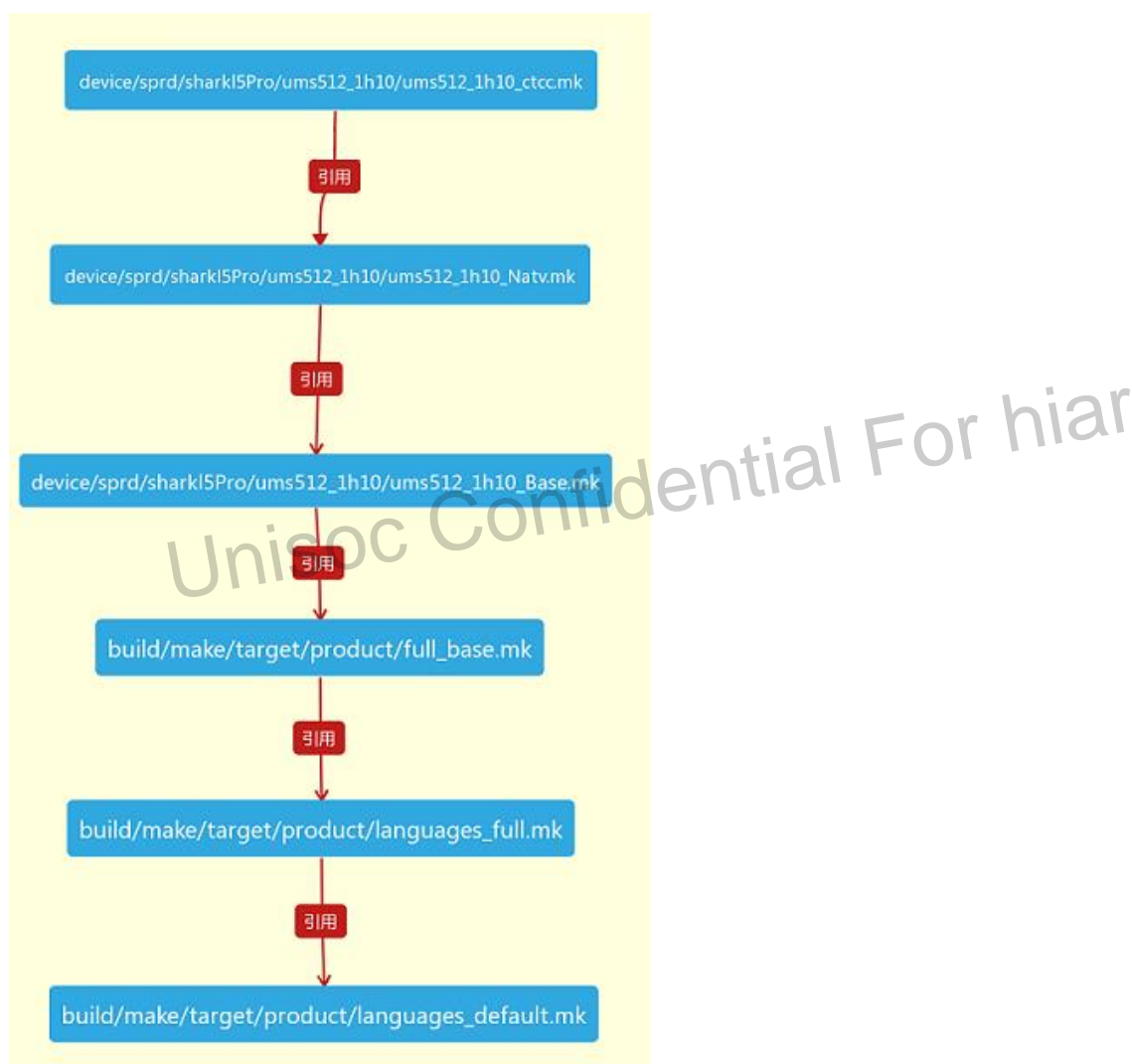
还有一些工程如 CTCC 工程的宏 **PRODUCT_REVISION** 没有配置 **multi-lang**，而是配置了运营商的名称 中国电信 **CTCC**，其指向的目录是 `vendor/sprd/feature_configs/carriers/ctcc`，而且 `FEATURES.PRODUCT_LOCALES` 属性在 `ctcc` 路径下的 `config.mk` 文件中没有配置，这种情况下系统支持的语言列表是 Android 系统原生 `PRODUCT_LOCALES` 属性配置语言，其配置路径如下：
`build/make/target/product/languages_default.mk`。

说明

加粗字体需要特别关注。

CTCC 工程 mk 文件引用该 mk 的路径如图 3-5 所示。

图3-5 CTCC 工程引用原生语言列表路径



3.3.3 配置默认语言

默认语言是指系统首次启动后生效的语言。默认语言可以有多种配置方法，具体如下：

- 宏 PRODUCT_LOCALES 或者 FEATURES.PRODUCT_LOCALES 配置的值中第一个语言通常会作为默认语言。如果两个宏都配置了值，根据宏配置的顺序，后配置的生效。
- 配置系统属性 ro.product.locale 的值，如果配置了该属性，则使用该系统属性的值作为默认语言，其优先级高于宏配置。
- 配置默认语言列表时，Settings.System.SYSTEM_LOCALES 配置项的第一个值将作为默认语言，优先级高于系统属性的配置或者宏的配置。
- 如果是 GMS 版本，有 Google 开机向导，则用户设置的语言将作为默认语言，其优先级最高。

关于 Settings.System.SYSTEM_LOCALES 配置项的配置，可以参考 3.3.1 配置默认语言列表，关于宏 PRODUCT_LOCALES 或者 FEATURES.PRODUCT_LOCALES 配置，可以参考 3.3.2 配置语言支持列表。

3.4 搜索相关问题

3.4.1 删除一个界面的搜索

如果需要删除整个功能界面的搜索，直接将类名上面的注解删除即可，如图 3-6 所示。

图3-6 标记 DisplaySettings 可搜索的注解

```
xref: /sprdroidr_trunk/packages/apps/Settings/src/com/android/settings/DisplaySettings.java
Home | History | Annotate | Line# | Scopes# | Navigate# | Raw | Download
@SearchIndexable(forTarget = SearchIndexable.ALL, @SearchIndexable.ARC)
public class DisplaySettings extends DashboardFragment implements OnFocusListenable {
    private static final String TAG = "DisplaySettings";
```

3.4.2 删除一个菜单的搜索

SearchIndexProvider 中实现 getNonIndexableKeys 方法，将对应菜单的 KEY 添加进去，如图 3-7 所示。

图3-7 删除一个菜单的搜索

```
@Override
public List<String> getNonIndexableKeys(Context context) {
    final List<String> keys = super.getNonIndexableKeys(context);
    keys.add(KEY_SWIPE_DOWN);
    return keys;
}
```

3.5 显示相关问题

3.5.1 Google 菜单的图标更换

3.5.1.1 Google 菜单

Google 菜单如图 3-8 所示。

图3-8 Google 菜单



3.5.1.2 图标更换

设置首界面每个菜单都是一个 Preference，数据来源是一个与 Preference 绑定 Tile 对象。在 Android 10.0 上，Tile 实现发生了变化，无法直接修改其中的 Icon 变量。因此要修改这个 Google 应用的图标，要在 Preference 与 Tile 绑定的时候，判断 Tile 是否是 Google 菜单，如果是就给 Preference 设置期望的图标。

绑定 Tile 和 Preference 的实现是在 DashboardFeatureProviderImpl.java 文件中，文件路径：
Settings/src/com/android/settings/dashboard/DashboardFeatureProviderImpl.java。

修改方法

在 bindIcon 方法的最后添加更换图标的代码：

```
ThreadUtils.postOnBackgroundThread() -> {
    String key = getDashboardKeyForTile(tile);
    if (!TextUtils.isEmpty(key)
        && key.endsWith("com.google.android.gms.app.settings.GoogleSettingsIALink")) {
        final Icon icon =
            Icon.createWithResource(mContext, R.drawable.ic_homepage_network);
        ThreadUtils.postOnMainThread() -> {
            preference.setIcon(icon.loadDrawable(preference.getContext()))
        };
    }
};
```

3.5.2 开放源代码许可定制

3.5.2.1 开放源代码许可说明

Android 10.0 显示开放源代码许可的菜单名称为【第三方许可】，从 Android 8.1 开始已经是这个标题。其显示的是 NOTICE.html 文件的内容，该文件在手机的/data/user_de/0/com.android.settings/cache 路径下。这个文件是由四个文件组合生成的，这四个文件分别是：

- /system/etc/NOTICE.xml.gz
- /vendor/etc/NOTICE.xml.gz
- /odm/etc/NOTICE.xml.gz
- /oem/etc/NOTICE.xml.gz

设置应用负责解析这些文件并组合生成最终的 NOTICE.html 文件。

3.5.2.2 开放源代码许可定制

NOTICE.html 最终是在设置应用的 LicenseHtmlGeneratorFromXml.java 中生成的。文件路径：
frameworks/base/packages/SettingsLib/src/com/android/settingslib/license/LicenseHtmlGeneratorFromXml.java

通常客户希望在许可内容的开头和结尾添加特定的声明内容。遇到这种情况，可以通过如下两个步骤进行修改：

步骤 1 文件 LicenseHtmlGeneratorFromXml 中添加两个常量，代表客户要求信息的开头和结尾字符串，下面代码可以作为示例：

```
private static final String HTML_CUSTOMER_HEADER_STRING =
    "<html><head>\n" +
    "<style type=\"text/css\">\n" +
    "body { padding: 0; font-family: sans-serif; }\n" +
    "same-license { background-color: #eeeeee;\n" +
    "border-top: 20px solid white;\n" +
    "padding: 10px; }\n" +
    "label { font-weight: bold; }\n" +
    "file-list { margin-left: 1em; color: blue; }\n" +
    "</style>\n" +
    "</head>" +
    "<body topmargin=\"0\" leftmargin=\"0\" rightmargin=\"0\" bottommargin=\"0\">\n" +
    "<div class=\"toc\">\n" +
    "<table cellpadding=\"0\" cellspacing=\"0\" border=\"0\">\n" +
    "<tr><td><!-- start-license --> \n" +
    "<pre class=\"license-text\">\n" +
    "客户自定义内容 \n" +
    "</pre><!-- license-text -->\n" +
    "</td></tr><!-- same-license -->\n" +
    "</table>\n" +
    "</div><!-- table of contents -->\n"
```

```
"<div class=\"toc\">\n" +
"<ul>";
private static final String HTML_CUSTOMER_REAR_STRING =
    "<tr><td><!-- start-license -->\n" +
    "<pre class=\"license-text\">\n" +
    "客户自定义内容 \n" +
    "</pre><!-- license-text -->\n" +
    "</td></tr><!-- same-license -->\n" +
    "</table></body></html>";
```

步骤 2 文件 LicenseHtmlGeneratorFromXml.java 的 **generateHtml** 方法中进行如下加粗字体表述修改：

```
@VisibleForTesting
static void generateHtml(Map<String, String> fileNameToContentIdMap,
    Map<String, String> contentIdToFileContentMap, PrintWriter writer,
    String noticeHeader) {
    List<String> fileNameList = new ArrayList();
    fileNameList.addAll(fileNameToContentIdMap.keySet());
    Collections.sort(fileNameList);
    writer.println(HTML_HEAD_STRING);
    //将HTML_HEAD_STRING替换为HTML_CUSTOMER_HEADER_STRING
    //中间省略部分代码
    writer.println(HTML_REAR_STRING);
    //将 HTML_REAR_STRING 替换为HTML_CUSTOMER_REAR_STRING
}
```

----结束

3.5.3 流量使用情况菜单变化

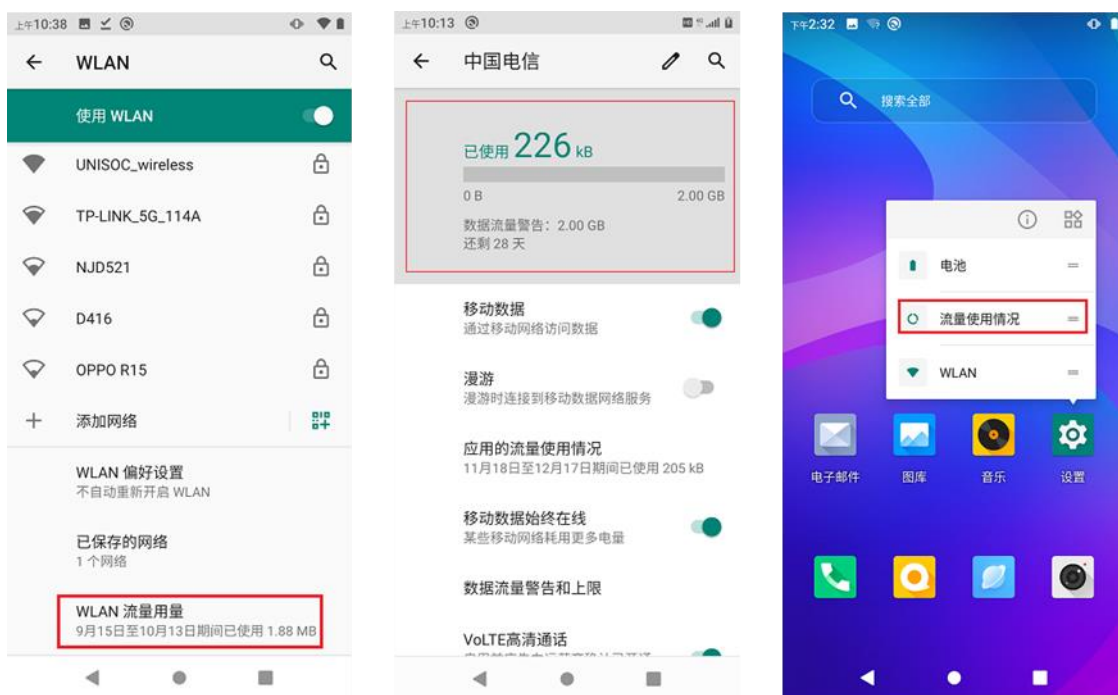
3.5.3.1 没有流量使用情况菜单

从 Android 10.0 版本开始，流量使用情况菜单已经移除，流量使用情况界面的功能被拆分，Wi-Fi 的流量使用情况移动到了 Wi-Fi 设置界面，移动网络的流量使用情况移动到了移动网络界面。

Android 10.0 并没有删除原本的流量使用情况界面，桌面长按设置应用图标，在弹出的快捷菜单框中，仍然保留了进入流量使用情况界面的入口。

具体如图 3-9 所示。

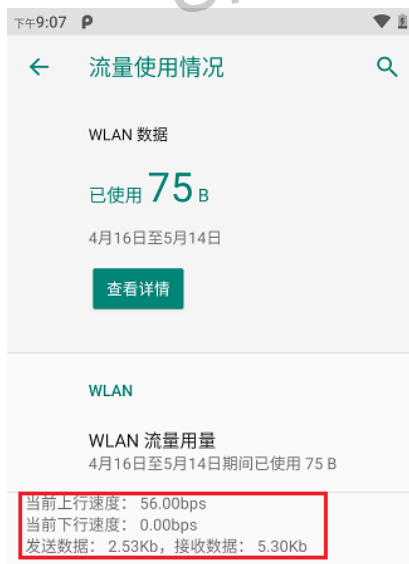
图3-9 流量使用情况功能



3.5.3.2 没有实时网络速率功能

Android 9.0 的流量使用情况界面有实时网络速率显示的功能，具体如图 3-10 所示。

图3-10 实时网络速率显示



说明

这是一个自研的功能，需求来源于运营商，目前运营商已没有此项要求，所以从 Android 10.0 开始，该功能已经废弃。

3.6 版本号定制相关问题

3.6.1 软件、硬件版本号定制

根据中国电信入库的需求，要求终端提供终端软件、硬件版本的查询。目前相关功能已导入到 Android 10.0 平台，但具体的软件版本号与硬件版本号需要客户自行配置。配置方法为：在中国电信版本对应的 Board 下新增一个 product.prop 文件，在其中配置两个系统属性值，具体的属性值需要客户自行定义。具体示例如图 3-11 所示。

图3-11 软件和硬件版本号配置

```
#Property info:software version
ro.version.software=L1693.6.01.01.00

#Property info:hardware version
ro.boot.hardware.revision=V1.00
```

配置了软件、硬件版本号属性的设备，在设置 → 关于手机界面会显示配置的软件版本号与硬件版本号。

3.6.2 内核版本号定制

内核版本号显示路径为：设置 → 关于手机 → Android 版本 → 内核版本。

设置中获取内核版本号的接口为：DeviceInfoUtils.getFormattedKernelVersion()，如图 3-12 所示。

文件路径：frameworks/base/packages/SettingsLib/src/com/android/settingslib/DeviceInfoUtils.java。

如果希望设置中显示定制的内核版本号，可以修改该接口的返回值。

图3-12 内核版本号获取接口

```
public static String getFormattedKernelVersion(Context context) {
    return formatKernelVersion(context, Os.uname());
}

@VisibleForTesting
static String formatKernelVersion(Context context, StructUtsname uname) {
    if (uname == null) {
        return context.getString(R.string.status_unavailable);
    }
    // Example:
    // 4.9.29-g958411d
    // #1 SMP PREEMPT Wed Jun 7 00:06:03 CST 2017
    final String VERSION_REGEX =
        "(#\\d+)" + /* group 1: "#1" */
        "(?\\.*)?" + /* ignore: optional SMP, PREEMPT, and any CONFIG_FLAGS */
        "((Sun|Mon|Tue|Wed|Thu|Fri|Sat).+)" + /* group 2: "Thu Jun 28 11:02:39 PDT 2012" */
    Matcher m = Pattern.compile(VERSION_REGEX).matcher(uname.version);
    if (!m.matches()) {
        Log.e(TAG, "Regex did not match on uname version " + uname.version);
        return context.getString(R.string.status_unavailable);
    }

    // Example output:
    // 4.9.29-g958411d
    // #1 Wed Jun 7 00:06:03 CST 2017
    return new StringBuilder().append(uname.release)
        .append("\n")
        .append(m.group(1))
        .append(" ")
        .append(m.group(2)).toString();
}
```

3.7 数据库相关问题

3.7.1 默认字体大小修改

1. 查看 packages/apps/Settings/res/values/arrays.xml，代码如下：

```
<string-array name="entries_font_size">
    <item msgid="6490061470416867723">Small</item>
    <item msgid="3579015730662088893">Default</item>
    <item msgid="1678068858001018666">Large</item>
    <item msgid="490158884605093126">Largest</item>
</string-array>

<string-array name="entryvalues_font_size" translatable="false">
    <item>0.85</item>
    <item>1.0</item>
    <item>1.15</item>
    <item>1.30</item>
</string-array>
```

从上面的逻辑可以知道，Small 与 0.85、Default 与 1.0、Large 与 1.15、Largest 与 1.30 一一对应。

2. 字体大小配置项是 Settings.System.FONT_SCALE，所以其在设置数据库的 System 表中。要修改其默认值，需要在 loadSystemSettings 方法中修改。

文件路径：SettingsProvider/src/com/android/providers/settings/DatabaseHelper.java

修改方法为添加如下代码：

```
// 1.30f对应最大字体，客户可自定义其他字体。  
loadSetting(stmt, Settings.System.FONT_SCALE, 1.30f);
```

3.7.2 默认关闭系统所有动画

1. 在 WindowManagerService 中将默认系统动画属性值置为 0，文件路径：

frameworks/base/services/core/java/com/android/server/wm/WindowManagerService.java

修改方法：找到如下代码位置：

```
private float mWindowAnimationScaleSetting = 1.0f;  
private float mTransitionAnimationScaleSetting = 1.0f;  
private float mAnimatorDurationScaleSetting = 1.0f;
```

将以上 3 个系统动画属性值修改为 “0.0f”：

```
private float mWindowAnimationScaleSetting = 0.0f;  
private float mTransitionAnimationScaleSetting = 0.0f;  
private float mAnimatorDurationScaleSetting = 0.0f;
```

2. 在设置数据库中，将动画的默认值设置为关闭，文件路径：

frameworks/base/packages/SettingsProvider/res/values/defaults.xml

修改方法：找到如下代码位置：

```
<fraction name="def_window_animation_scale">100%</fraction>  
<fraction name="def_window_transition_scale">100%</fraction>
```

将 “def_window_animation_scale” 和 “def_window_transition_scale” 的值修改为 “0%”：

```
<fraction name="def_window_animation_scale">0%</fraction>  
<fraction name="def_window_transition_scale">0%</fraction>
```

3.7.3 默认屏幕亮度修改

Android 9.0 之前通过配置 def_screen_brightness 的值，可以直接计算出默认亮度百分比。Android 9.0 开始将 def_screen_brightness 的值做了 gamma 曲线的转换，因此无法直接将其值计算为亮度百分比。所以如果客户需要设置特定百分比（例如 80%）的默认亮度，需要经过如下几个步骤：

- 步骤 1 代码中添加 log，拖动亮度条，当 log 中百分比显示为 80% 时，查看 log 中对应的亮度值是多少。具体添加方法如图 3-13 所示。

图3-13 默认屏幕亮度添加 log

```
diff --git a/src/com/android/settings/display/BrightnessLevelPreferenceController.java b/src/com/andro
index 4bb0a99..eab1919 100644
--- a/src/com/android/settings/display/BrightnessLevelPreferenceController.java
+++ b/src/com/android/settings/display/BrightnessLevelPreferenceController.java
@@ -137,6 +137,8 @@ public class BrightnessLevelPreferenceController extends AbstractPreferenceContr
     mMinBrightness, mMaxBrightness);
 }
+    Log.e(TAG, " SCREEN_BRIGHTNESS = " + Settings.System.getInt(mContentResolver,
+    System.SCREEN_BRIGHTNESS, mMinBrightness));
+    return getPercentage(value, 0, GAMMA_SPACE_MAX);
}
@@ -147,6 +149,7 @@ public class BrightnessLevelPreferenceController extends AbstractPreferenceContr
    if (value < min) {
        return 0.0;
    }
+    Log.e(TAG, " Percentage = " + (value - min) / (max - min));
    return (value - min) / (max - min);
}
```

步骤 2 将 log 打印出来的亮度值设置到 def_screen_brightness 中即可，文件路径：

frameworks/base/packages/SettingsProvider/res/values/defaults.xml

----结束

3.7.4 默认关闭快速打开相机

在 Android 10.0 上，快速打开相机菜单会有两种情况，可通过菜单的说明确认：

通过按音量上键两次快速打开相机

如果是双击音量上键开启相机，要默认关闭该功能，需要配置如下配置项的默认值。

配置项：Settings.Secure.CAMERA_DOUBLE_TAP_VOLUMEUP_GESTURE_DISABLED

文件路径：SettingsProvider/src/com/android/providers/settings/DatabaseHelper.java

修改方法：该配置项在 Secure 表中，要在 loadSecureSettings 方法中添加如下代码，配置一下默认值：0 为默认开启，1 为默认关闭。

```
loadIntegerSetting(stmt,
Settings.Secure.CAMERA_DOUBLE_TAP_VOLUMEUP_GESTURE_DISABLED, 1);
```

通过按电源键两次快速打开相机

如果是双击电源键开启相机，要默认关闭该功能，需要配置如下配置项的默认值。

配置项：Settings.Secure.CAMERA_DOUBLE_TAP_POWER_GESTURE_DISABLED

文件路径：SettingsProvider/src/com/android/providers/settings/DatabaseHelper.java

修改方法：该配置项在 Secure 表中，要在 loadSecureSettings 方法中添加如下代码，配置一下默认值：0 为默认开启，1 为默认关闭。

```
loadIntegerSetting(stmt,
Settings.Secure.CAMERA_DOUBLE_TAP_POWER_GESTURE_DISABLED, 1);
```

3.7.5 默认开启拿起手机即显示

拿起手机即显示即 Lift to check phone 功能，如果需要默认开启该功能，需要更改两个配置项的默认值，配置项：Settings.Secure.DOZE_PICK_UP_GESTURE 和 Settings.Secure.DOZE_ENABLED。

文件路径：frameworks/base/packages/SettingsProvider/res/values/defaults.xml

修改方法：找到如下代码位置：

```
<!-- Default for Settings.Secure.DOZE_PICK_UP_GESTURE -->
<bool name="def_doze_pick_up_gesture_enabled">false</bool>

<!-- Default for Settings.Secure.DOZE_ENABLED -->
<bool name="def_doze_enabled">false</bool>
```

将 “def_doze_pick_up_gesture_enabled” 和 “def_doze_enabled” 值修改为 “true”：

```
<!-- Default for Settings.Secure.DOZE_PICK_UP_GESTURE -->
<bool name="def_doze_pick_up_gesture_enabled">true</bool>

<!-- Default for Settings.Secure.DOZE_ENABLED -->
<bool name="def_doze_enabled">true</bool>
```

3.7.6 默认关闭始终开启移动数据网络

始终开启移动数据网络是开发者选项界面中 Mobile data always active 功能，这个功能默认打开，当功能打开时，Wi-Fi 网络可以和移动网络共存。如果需要默认关闭这个功能，可以修改这个功能在数据库中对配置项的默认值。

配置项：Settings.Global.MOBILE_DATA_ALWAYS_ON。

文件路径：frameworks/base/packages/SettingsProvider/res/values/defaults.xml

修改方法：找到如下代码位置：

```
<!-- Default setting for Settings.Global.MOBILE_DATA_ALWAYS_ON -->
- <bool name="def_mobile_data_always_on">true</bool>
```

将 “def_mobile_data_always_on” 值修改为 “false”：

```
<!-- Default setting for Settings.Global.MOBILE_DATA_ALWAYS_ON -->
<bool name="def_mobile_data_always_on">false</bool>
```

3.7.7 配置第三方输入法为系统默认输入法

要配置第三方输入法为系统默认输入法，需要几个步骤，下面以配置搜狗输入法为示例，说明配置默认输入法的步骤：

步骤 1 将搜狗输入法预置到系统的 system/vital-app 路径下，预置应用的方法此处不做详述。

步骤 2 在预置了搜狗输入法的设备上设置手机输入法：设置 → 语言和输入法 → 虚拟键盘，打开搜狗输入法，并将搜狗输入法做为默认输入法。

步骤 3 记录设置数据库 Secure 表中 DEFAULT_INPUT_METHOD 和 ENABLED_INPUT_METHODS 两个配置项对应的内容。可以通过如下命令进行获取：

```
adb shell settings get secure default_input_method //获取 DEFAULT_INPUT_METHOD
adb shell settings get secure enabled_input_methods //获取 ENABLED_INPUT_METHODS
```

步骤 4 定义 DEFAULT_INPUT_METHOD 和 ENABLED_INPUT_METHODS 的默认值

在 frameworks/base/packages/SettingsProvider/res/values/defaults.xml 中添加两个默认值配置项，用于定义默认值，默认值为步骤 3 中记录的搜狗输入法的值，具体如图 3-14 所示。

图3-14 默认输入法配置

```
<!-- Default for Settings.Secure.DEFAULT_INPUT_METHOD -->
<string name="def_default_input_method" translatable="false">com.sohu.inputmethod.sogou/.SogouIME</string>

<!-- Default for Settings.Secure.ENABLED_INPUT_METHODS -->
<string name="def_enabled_input_methods" translatable="false">com.sohu.inputmethod.sogou/.SogouIME</string>
```

步骤 5 将默认值加载到设置的数据库中

因为默认输入法的两个配置项都在 Secure 表中，所以加载默认值在 DatabaseHelper.java 文件的 loadSecureSettings 方法中修改。

文件路径：SettingsProvider/src/com/android/providers/settings/DatabaseHelper.java

修改方法如图 3-15 所示。

图3-15 加载默认输入法到数据库

```
/* Set default input method @{ */
String defaultInput = mContext.getResources().getString(R.string.def_default_input_method);
if (!TextUtils.isEmpty(defaultInput)) {
    loadSetting(stmt, Settings.Secure.DEFAULT_INPUT_METHOD, defaultInput);
}
String enabledInputs = mContext.getResources().getString(R.string.def_enabled_input_methods);
if (!TextUtils.isEmpty(enabledInputs)) {
    loadSetting(stmt, Settings.Secure.ENABLED_INPUT_METHODS, enabledInputs);
}
/* @} */
```

----结束