Android PowerHal
场景管理开发指南

版本号: 1.2
发布日期: 2025.4.9

版本历史

版本号	日期	制/修订人	内容描述
1.0	2023.12.11	AWA1696	初版开发指南。
1.1	2024.11.11	AWA1696	1. 更新 sysdir 支持； 2. 新增 uevent 使用方法； 3. 新增 dumpsys 和替换配置文件方法； 4. 更新新平台适配章节； 5. 增加优先级制定规格章节。
1.2	2025.4.9	AWA1696	1. 更新 read 配置支持； 2. 更新 demo 使用示例；

目　录

插　图

1 概述

1.1 编写目的

本文档主要描述 Android PowerHal 场景管理如何进行开发和维护，以及相关开发人员如何调用和调试场景管理。

1.2 适用范围

适用于 Android13 及之后的 Android 平台。

1.3 相关人员

1. PowerHal 模块维护工程师；
2. 需要对 PowerHal 场景管理有需求的开发工程师。

2 模块介绍

PowerHal 是 Android 原生场景管理模块，提供多种 Mode 和 Boost，达到最优化性能和功耗的目的。

Aw 在原生的 PowerHal 的基础上，增加了 AeExtensionPower 扩展接口和 UeventMonitor 拓展模式，可从多个位置触发设置对应的场景。

当前 PowerHal 支持功能如下：

功能接口	功能说明
setMode	原生接口，支持
setBoost	原生接口，支持
setExtensionBoost	Aw 拓展接口，用于设置自定义的 scene
getResourceValue	Aw 拓展接口，用于获取指定资源的值，便于监控系统状态
UeventMonitor	Aw 拓展接口，支持通过 Uevent 设置模式

2.1 模块架构

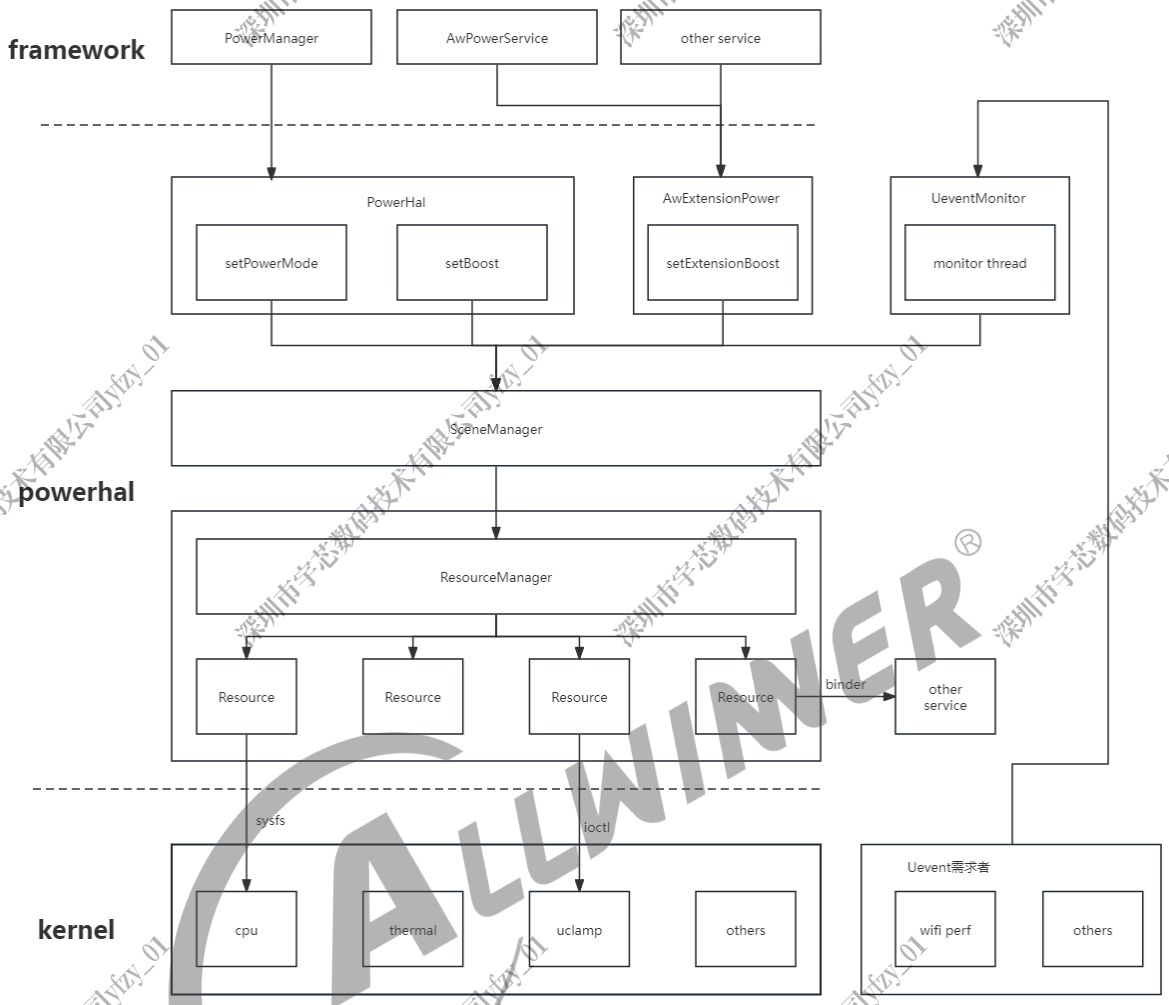

图 2-1: powerhal 序列化架构

2.2 架构说明

PowerHal 分为 3 层架构，分为入口层，SceneManager 以及 ResourceManager

场景入口

1. PowerHal 可通过原生接口 setMode 和 setBoost 设置场景；
2. AwExtensionPower 为扩展服务，可通过 setExtensionBoost 设置场景和 getResourceValue 获取预配置的资源值；
3. UeventMonitor 可捕获内核层上报的 uevent 并设置对应的场景。

SceneManager 及 ResourceManager

1. 所有场景统一由 SceneManager 进行管理，并且所有场景的调用都只有 SceneManager 一个入口；
2. ResourceManger 负责管理所有的 Resource，并受 SceneManager 的控制；
3. Resource 控制对应实际的资源，如 cpu governor、ddrfreq、IO 等内核资源，也可通过 binder 调用其他进程的方法作为资源控制。

2.3 源码路径及源码结构

当前使用 PowerHal 场景管理代码路径如下。

```
hardware/aw/power/aidl
```

整体源码结构如下。

```
.
├── configs                          //场景配置文件
│   ├── a333
│   ├── config.mk
│   ├── default
│   ├── sun50iw10p1
│   ├── sun55iw3p1
│   ├── sun60iw2p1
│   └── sun65iw1p1
├── demo                             //demo使用例子，供参考
│   ├── cpp
│   ├── java
│   └── ndk
├── impl                             //powerhal核心代码逻辑
│   ├── Android.bp
│   ├── bsp
│   ├── include
│   ├── libawpowergen.h.template
│   ├── Power.cpp
│   ├── PowerFactory.cpp
│   ├── powerhint
│   ├── Resource.cpp
│   ├── ResourceManager.cpp
│   ├── Scene.cpp
│   ├── SceneManager.cpp
│   ├── SysDirResource.cpp
│   ├── UclampLimitResource.cpp
│   ├── UeventMonitor.cpp
│   └── Utils.cpp
├── libperfmgr                       //原生adpf接口
├── libs                             //demo ndk库，用于直接调用AwExtendtion拓展接口
│   └── ndk
└── service                          //服务启动代码
```

3 场景和资源说明

3.1 Scene

3.1.1 定义

首先要定义什么是 scene，一个场景表示当前设备正在处于某个状态；

如开机启动，可定义为 booting；安兔兔跑分也是一个场景，定义为 performance；

持续性

不同场景，可能存在不同的持续时间，如开机启动，在开完机的时候结束；音乐播放，要退出音乐场景为止。

可叠加性

多个场景可以同时存在，同时控制，如音乐播放场景时叠加游戏场景。

3.1.2 功能

当进入某个场景时，请求指定的资源，从而达到特定的性能，达到最优的体验。

如在开机启动场景，需要 CPU、DDR、IO 达到最大性能，以达到最快的开机启动速度。

3.1.3 Scene 配置

Scene 配置需要在定义的场景中，对存在的 Resource 进行控制，当前一份策略参考如下。

```
<scene name="INTERACTIVE">
    <set resource="cpu_governor" value="schedutil"/>
    <set resource="dfs_governor" value="sunxi_actmon"/>
    <set resource="io_schedule" value="mq_deadline_low_latency"/>
    <set resource="thermal_control" value="enabled"/>
    <set resource="nsi_limit" value="limit"/>
</scene>
```

图 3-1: scene 配置

解析如下：

- name：场景名称
- set：该场景下，需要进行调控的资源。

那么在 INTERACTIVE 场景下，其完成的资源控制有：

- cpu 策略：调频模式；
- ddr 策略：调频模式；
- IO 策略：指定的配置；
- thermal：打开温控；
- nsi 带宽限制：打开。

可以看到，一个场景他是可以控制多份资源的配置的，那 Scene 中配置的资源又是如何定义的，多个场景需要控制同一份资源的时候该怎么办？继续 Resource 类的说明。

说明

原生的 Mode 和 Boost 也是一种 scene，并需要注意使用原生的 Mode 和 Boost 时，scene 的 name 应保持一致。

3.2 Resource

3.2.1 定义

在 PowerHal 中，将所有可调控的资源定义为 resource，如一个 sysfs 的文件节点，一个 ioctl 调用等。

当前常见的几种资源类型：

- sysfs：系统节点，可直接读写；
- ioctl：与 sysfs 读写类似，但存在自定义参数，故需特殊处理；
- syscall：系统调用，每个系统调用类型的参数均不一样，需特殊处理；
- binder_call：调用其他进程服务，进行资源管控；
- sysdir：支持通过匹配节点值去匹配可变路径的节点，匹配成功后相当于一个 sysfs。

3.2.2 功能

一个 Resource 需要支持的功能如下：

1. 对资源直接进行调控；

2. 定义不同配置的优先级；
3. 在多个请求配置时，采用最高优先级配置；
4. 根据请求配置的 timeout 时间，设置计时器，在超时到达后，清除对应的请求；
5. 当无请求存在时，恢复到默认配置。

3.2.2.1 timeout 时间说明

当请求 resource 时，timeout 可设置为如下 3 个值：

- 0：表示常开，永不超时；
- -1：表示退出，结束该请求配置；
- 大于 0：超时时间，表示请求该配置多长时间；单位 ms；最低 500ms，低于 500ms 将被修正为 500ms。

3.2.3 配置

当前架构中，Resource 的一份资源配置如下：

```
<resource name="cpu_governor" type="sysfs" default="schedutil">
    <config name="performance"  priority="1">
        <write path="/sys/devices/system/cpu/cpufreq/policy0/scaling_governor" value="performance"/>
        <write path="/sys/devices/system/cpu/cpufreq/policy4/scaling_governor" value="performance"/>
        <write path="/sys/devices/system/cpu/cpufreq/policy6/scaling_governor" value="performance"/>
    </config>
    <config name="schedutil"  priority="2">
        <write path="/sys/devices/system/cpu/cpufreq/policy0/scaling_governor" value="schedutil"/>
        <write path="/sys/devices/system/cpu/cpufreq/policy4/scaling_governor" value="schedutil"/>
        <write path="/sys/devices/system/cpu/cpufreq/policy6/scaling_governor" value="schedutil"/>
        <write path="/sys/devices/system/cpu/cpufreq/schedutil/rate_limit_us" value="20000" />
    </config>
    <read name="cpub_governor" path="/sys/devices/system/cpu/cpufreq/policy6/scaling_governor"/>
</resource>
```

图 3-2: cpu-governor

如图所示，上图是一份通用的 cpu 策略的配置，参数说明如下：

resource 节点

- name：资源的名称，用于 scene 中进行控制；
- type：资源的类型，这里是 sysfs，表示通过节点读写方式；
- default：默认值：当该资源没有调用者时，默认使用的配置。

config 节点

预定义的配置，每个配置的优先级和需要进行的操作：

- name：配置名称，用于 scene 中进行控制；

- priority：优先级配置，数字小的优先级高。

write 节点：path 表示 sysfs 中需要读写的路径，value 为需要写的值。

read 节点：表示支持读取的 resource 信息，path 为需要读取的路径的值。

另一份 syscall 的资源配置参考如下。

```
<resource name="uclamp_limit" type="syscall" default="unlimit">
    <config name="unlimit"  priority="1">
        <write proc="rcu_preempt" thread="rcu_preempt" min="1024" max="1024"/>
        <write proc="/system/bin/lmkd" thread="lmkd" min="1024" max="1024"/>
        <write proc="/vendor/bin/hw/android.hardware.graphics.composer-service.sunxi" thread="CommitThread" min="1024" max="1024"/>
    </config>
    <config name="limit"  priority="2">
        <write proc="rcu_preempt" thread="rcu_preempt" min="0" max="1024"/>
        <write proc="/system/bin/lmkd" thread="lmkd" min="0" max="1024"/>
        <write proc="/vendor/bin/hw/android.hardware.graphics.composer-service.sunxi" thread="CommitThread" min="0" max="1024"/>
    </config>
</resource>
```

图 3-3: syscall 资源定义

这里使用的 syscall 为 uclamp，主要用于限制 uclamp，故其 write 值也不一致，解析也存在差异。

3.2.3.1 read 配置详解

当前 PowerHal 支持读取预配置的信息，用于用户空间获取一些设备的实时信息，监控设备状态。

一份标准的 read 的 resource 配置如下。

```
<resource name="cpu_max_freq" type="sysfs">
    <read name="cpul_max_freq" path="/sys/devices/system/cpu/cpufreq/policy0/scaling_max_freq"/>
    <read name="cpum_max_freq" path="/sys/devices/system/cpu/cpufreq/policy4/scaling_max_freq"/>
    <read name="cpub_max_freq" path="/sys/devices/system/cpu/cpufreq/policy6/scaling_max_freq"/>
</resource>
```

图 3-4: read 配置

这里需注意如下几点。

1. 当一个 resource 中只存在 read 类型配置时，可以不写 default config；
2. resource name 为读取的标志，通过接口 getResourceValue 读取时，会一次性读取所有的 read 配置；
3. 读取数据返回值为 [name, value] 格式的数组，其中 name 为 read 配置中的 name。

⚠ 注意

当前仅 sysfs 类型 resource 支持 read 信息。

3.2.4 多场景共存

场景设置不是单一的，同一时间下，可能会有多个场景存在，比如我们可以一边听歌一边看小说。

那同样的，一份资源，也会同时存在多个场景的请求，并且每个场景的请求的配置可能还是不一样的，这时候如何做判断使用哪个配置呢?

那这里就需要使用到配置优先级的定义了。

如下图所示。

```
ResourceId = 0x2 ResoureceName: dfs_governor
  mResourceType=sysfs mDefaultConfig=sunxi_actmon
  mCurrentConfig=performance

  Config1 ConfigName=performance priority=1
    Write1<TargetPath=/sys/class/devfreq/3120000.dmcfreq/governor Value=performance>
  Config2 ConfigName=sunxi_actmon priority=2
    Write1<TargetPath=/sys/class/devfreq/3120000.dmcfreq/governor Value=sunxi_actmon>

  mCurrentItem = [mCallerId=0 mSceneId=0x00000005 mConfigName=performance mRequestTimeMs=0]
  All Items :
    Item1 : [mCallerId=0 mSceneId=0x00000007 mConfigName=sunxi_actmon mRequestTimeMs=0]
    Item2 : [mCallerId=0 mSceneId=0x00000005 mConfigName=performance mRequestTimeMs=0]
```

图 3-5: 多场景共存

从图中可以看出。

1. 当前资源为 dfs_governor，资源类型为 sysfs，默认的 Config 配置为 sunxi_actmon，当前的配置为 performance；
2. 支持 2 个 Config，分别为 performance 和 sunxi_actmon，其中 performance 优先级为 1，sunxi_actmon 优先级为 2，前面有提到，数字越小，优先级越高。故 performance 配置的优先级应是高于 sunxi_actmon 的；
3. 从 All Items 中可以看到，当前存在了 2 个场景，并且同时请求了不同的配置，而此时从第 2 点可以得出，当多场景同时请求时，响应高优先级的配置，故此时可以看到 mCurrentItem 是响应了高优先级 performance 的请求；
4. 可以通过查看 mCurrentConfig，mCurrentItem 以及 All Items 里的配置优先级是否一致且符合最高优先级判断，来查看是否存在异常。

3.2.4.1 优先级制定规则

一个 resource 可以有多个 config 配置，且每个 config 的优先级配置不一，下面简单说明一下该如何去制定不同 config 之间的优先级。

性能优先原则

针对影响性能的 resource，即高性能场景，优于低性能场景，该原则主要目的是为了优先保障用户体验，以 cpu_governor 为例，performance 优先级大于 schedule 来确保进入 performance 时能发挥出最大的性能。

功耗优先原则

针对部分影响功耗的 resource，出于节省功耗的目的，低功耗 config 优先级通常大于高功耗 config，但由于性能和功耗通常是矛盾的，二者不能兼得，故针对这种情况时，需要结合实际情况来分析。

自定义优先级

部分配置，往往需要基于实际的场景除非去配置优先级，以温控为例，当前存在温控开和关 2 个配置。

```
<resource name="thermal_control" type="sysdir" default="enabled">
    <config name="disabled"  priority="1">
        <write dir="/sys/class/thermal" read_node="type" read_value="cpul_thermal_zone" write_node="mode"
 value="disabled"/>
    </config>
    <config name="enabled"  priority="2">
        <write dir="/sys/class/thermal" read_node="type" read_value="cpul_thermal_zone" write_node="mode"
 value="enabled"/>
    </config>
</resource>
```

一般来说，关闭温控，可能会导致机器过热，甚至损坏机器，那么出于保护机器的目的，使能温控优先级应高于关闭温控。

但结合实际情况：温控通常是常开的，特殊场景下（如跑分）需要关闭温控，来达到最大的性能，而这种场景一般是比较少且固定的，那这种情况下关闭温控的优先级应高于打开温控，同时在场景的使用上要尽可能的细分，避免温控常关导致过热异常。

4 新平台适配

针对新增的 IC 或者平台，则需增加一份配置文件，定义好 Resource，以及相关的 Scene-Resource 即可，无需修改代码。

配置文件路径：

```
/hardware/aw/power/aidl/configs
```

配置文件列表

```
.
├── config.mk                          //编译控制文件
├── default                            //默认配置
├── a333                               //a333平台，通过TARGET_BOARD_IC获取
│   ├── aw_power_resource.xml
│   └── aw_power_scene_config.xml
├── sun50iw10p1
│   ├── aw_power_resource.xml
│   └── aw_power_scene_config.xml
├── sun55iw3p1
│   ├── aw_power_resource.xml
│   └── aw_power_scene_config.xml
├── sun60iw2p1                         //sun60iw2p1平台，通过TARGET_BOARD_CHIP获取
│   ├── aw_power_resource.xml
│   └── aw_power_scene_config.xml
└── sun65iw1p1                         //sun65iw1p1平台，通过TARGET_BOARD_CHIP获取
    ├── aw_power_resource.xml
    └── aw_power_scene_config.xml
```

4.1 平台配置匹配顺序

在编译过程中，会把需要的配置文件拷贝到 vendor/etc 路径下，针对不同方案不同 IC，可能会有不同的需求，不同版型使用的配置由于场景的差异而需要使用不同的配置，故定义的配置原则是如下：

1. 如果方案中指定了配置，则优先使用配置文件；
2. 如果方案中未指定某一配置，则匹配PRODUCT_DEVICE，如 a523-pro；
3. 如果 2 未成功匹配，则匹配TARGET_BOARD_IC，如 a733/a523/a133；
4. 如果 3 未成功匹配，则匹配TARGET_BOARD_CHIP，如 sun50iw10p1；
5. 如果 3 还是未找到，则使用 default，同时编译时会有警告 log，异常报错 log 如下。

```
hardware/aw/power/aidl/configs/config.mk:12: warning: powerhal use default config:hardware/aw/power/aidl/configs/
    default, you need to make some adaptations
```

配置文件匹配优先级：指定配置 > PRODUCT_DEVICE > TARGET_BOARD_IC > TARGET_BOARD_CHIP > default。

技巧

获取变量PRODUCT_DEVICE、TARGET_BOARD_IC、TARGET_BOARD_CHIP可以在执行完 source ./build/envsetup.sh 和 lunch 后，通过命令get_build_var来获取

⚠ 注意

default 配置通常不应该被使用，主要是为了解决编译报错和无法启动所设置的，如项目中遇到使用 default 配置，需要尽快刷新配置文件。

4.2 指定配置文件

在方案中，设置变量PRODUCT_SPECIAL_POWERCONFIG为 true，则自动匹配配置文件功能将关闭，需要开发人员手动复制配置文件到 vendor/etc 路径，使用例子如下。

1. 将配置文件放入到方案目录路径下，例如 saturn/${PRODUCT_DEVICE}/system/路径下。
2. 在 saturn/${PRODUCT_DEVICE}/system/config.mk 中，增加如下命令。

```
PRODUCT_SPECIAL_POWERCONFIG := true

PRODUCT_COPY_FILES += \
    $(LOCAL_MODULE_PATH)/aw_power_resource.xml:$(TARGET_COPY_OUT_VENDOR)/etc/aw_power_resource.xml \
    $(LOCAL_MODULE_PATH)/aw_power_scene_config.xml:$(TARGET_COPY_OUT_VENDOR)/etc/
      aw_power_scene_config.xml
```

上述命令表示，powerhal 将使用当前路径下的配置文件。

并且编译刚开始时，可看到如下特殊打印。

```
hardware/aw/power/aidl/configs/config.mk:20: warning: power hal use special config
```

有上述打印就表示使用了特殊配置文件。

⚠ 注意

使用特殊配置文件时，需要确保正确的将 2 份配置文件编译到了 vendor/etc 目录，否则 PowerHal 可能无法正常工作。

4.3 使用方法

4.3.1 新增 IC

如新增一个TARGET_BOARD_CHIP平台支持，如 sunXXiwXpX，此时可通过命令get_build_var先获取TARGET_BOARD_CHIP的值，然后在配置文件目录中，新建一个文件夹。

参考命令。

```
get_build_var TARGET_BOARD_CHIP
```

说明

需要先执行 source ./build/envsetup.sh 和 lunch 对应的方案后，再运行。

配置文件目录路径

```
/hardware/aw/power/aidl/configs
```

例如。

```
/hardware/aw/power/aidl/configs/sunXXiwXpX
```

然后可以将另一相似平台的配置文件，拷贝到该文件，并针对 sunXXiwXpX 平台进行新的配置修改，即可。

完成后，如下所示。

```
.
├── config.mk                          //编译控制文件
├── default                            //默认配置
│   ├── aw_power_resource.xml
│   └── aw_power_scene_config.xml
└── sunXXiwXpX                         //sunXXiwXpX平台，通过TARGET_BOARD_CHIP获取
    ├── aw_power_resource.xml
    └── aw_power_scene_config.xml
```

编译，启动后查看是否生效。

4.3.2 新增特定设备

在上一章节，我们新增了 sunXXiwXpX 的配置，sunXXiwXpX 有多个 PRODUCT_DEVICE，此时需要针对某一 PRODUCT_DEVICE 使用特殊的配置文件，则可参考下列步骤进行适配。

获取 PRODUCT_DEVICE 的值

通过命令get_build_var获取对应的值，如 a5xx-pro;

创建新的配置

在如下路径中，创建 a5xx-pro 文件夹

```
/hardware/aw/power/aidl/configs
```

创建完成后

```
/hardware/aw/power/aidl/configs/a5xx-pro
```

并将 sunXXiwXpX 的平台配置拷贝到该文件夹里，同时针对 a5xx-pro 进行修改，完成后如下。

```
.
├── config.mk                          //编译控制文件
├── default                            //默认配置
│   ├── aw_power_resource.xml
│   └── aw_power_scene_config.xml
├── a5xx-pro                           //a5xx-pro配置，通过PRODUCT_DEVICE获取
│   ├── aw_power_resource.xml
│   └── aw_power_scene_config.xml
└── sunXXiwXpX                         //sunXXiwXpX平台，通过TARGET_BOARD_CHIP获取
    ├── aw_power_resource.xml
    └── aw_power_scene_config.xml
```

编译，启动后查看是否生效。

5 新增场景和资源

对以后可能出现的新 IP 中的新功能，用何种方式支持。(增加接口/使用类似 ioctrl 的方式增加入参枚举类型的数量/结构体新增成员, 等等)。

5.1 新增 Scene

5.1.1 增加 secene config

在对应版型所使用的 sceneconfig 文件中，增加对应的场景和所需求的配置，比如 face_unlock 希望可以提高系统 CPU 和 DDR 频率来加快人脸解锁。

```
<scene name="face_unlock">
    <set resource="dfs_governor" value="performance"/>
    <set resource="cpu_governor" value="performance"/>
</scene>
```

配置完成后，可以先通过 dumpsys 接口来进行调试，看是否满足需求，如满足，则可以在对应场景进入和退出的位置增加相关的调用即可。

5.2 新增 Resource 说明

5.2.1 新增 SysfsResource

新增一个 sysfs 形式的 Resource，只需要直接添加到对应的 Resource 文件以及相关的 Scene-Resource 即可。

```
<resource name="cpu_governor" type="sysfs" default="schedutil">
    <config name="performance"  priority="1">
        <write path="/sys/devices/system/cpu/cpufreq/policy0/scaling_governor" value="performance"/>
        <write path="/sys/devices/system/cpu/cpufreq/policy4/scaling_governor" value="performance"/>
        <write path="/sys/devices/system/cpu/cpufreq/policy6/scaling_governor" value="performance"/>
    </config>
    <config name="schedutil"  priority="2">
        <write path="/sys/devices/system/cpu/cpufreq/policy0/scaling_governor" value="schedutil"/>
        <write path="/sys/devices/system/cpu/cpufreq/policy4/scaling_governor" value="schedutil"/>
        <write path="/sys/devices/system/cpu/cpufreq/policy6/scaling_governor" value="schedutil"/>
        <write path="/sys/devices/system/cpu/cpufreq/schedutil/rate_limit_us" value="20000" />
    </config>
    <read name="cpub_governor" path="/sys/devices/system/cpu/cpufreq/policy6/scaling_governor"/>
</resource>
```

图 5-1: cpu-governor

5.2.2 新增 SyscallResource 或者 IoctlResource

以 Uclamp 控制，说明如何新增 syscall 类型的 resource，如下是 Uclamp 的实现。

1. 新建 UclampLimitResource 类，其继承于 Resource。

```
class UclampLimitResource : public Resource {
  public:
    UclampLimitResource(xmlNodePtr node);

  private:
    void setConfig(Config* config) override;
    void init();

    Config* parseConfigByNode(xmlNodePtr node) override;
}; //class UclampLimitResource
```

图 5-2: UclampLimitResource

其中必须实现 setConfig 和 parseConfigByNode 两个方法。

- setConfig：setConfig 实现 syscall 的具体调用；
- parseConfigByNode：解析调用 syscall 时所需要的所有参数。

2. 由于 config 都是使用 write 指针进行存储的，故还应实现一个特定的 UclampWrite 类继承于 Write，其中存储所需要的配置文件。

```
class UclampWrite : public Write {
  public:
    UclampWrite();

    std::string toString() override;

    std::string mProc;
    std::string mThread;
    int mTid;
    int mPid;
    int mMin;
    int mMax;
}; //class UclampWrite
```

图 5-3: UclampWrite

可以看到 uclamp 需要存储进程名，线程名，以及对应需要配置的 min 和 max，同时 uclamp 是通过 tid 来控制的，故还需要存储一下 pid 和 tid。

3. 在 PowerConfig.h 中，增加对应的 syscall 类型。

```
typedef enum {
    UNKNOWN_SYSCALL = -1,
    UCLAMP_LIMIT = 0,
} SyscallResourceType;
```

图 5-4: SyscallResourceType

4. 在 Util.cpp 中，完善转换函数。

```
std::string syscallResourceTypeToString(SyscallResourceType type) {
    switch (type) {
        case UCLAMP_LIMIT:
            return std::string("uclamp_limit");
        default:
            return std::string("unknown");
    }
}
SyscallResourceType syscallResourceStringToType(std::string resourceString) {
    if (resourceString == std::string("uclamp_limit")) {
        return SyscallResourceType::UCLAMP_LIMIT;
    }
    return SyscallResourceType::UNKNOWN_SYSCALL;
}
```

图 5-5: ResourceStringToType

5. 在 ResourceManager 的 parseSysCallResourceByNode，增加对应的分支情况，并在对应分支下，通过 new 的方式获取一个新的 UclampLimitResource 指针。

```
Resource* ResourceManager::parseSysCallResourceByNode(xmlNodePtr node) {
    xmlChar *resourcename = NULL;
    std::string syscallName;
    SyscallResourceType sysCallType;
    Resource* resource = nullptr;

    resourcename = xmlGetProp(node, (const xmlChar*) "name");
    syscallName = std::string(reinterpret_cast<const char*>(resourcename));
    xmlFree(resourcename);

    sysCallType = syscallResourceStringToType(syscallName);

    switch (sysCallType) {
        case SyscallResourceType::UCLAMP_LIMIT:
            resource = new UclampLimitResource(node);
            break;
        default:
            resource = nullptr;
            ALOGD("%s %d", __func__, __LINE__);
            break;
    }
    return resource;
}
```

图 5-6: parseSysCallResourceByNode

6. 在 UclampLimitResource 中完善构造函数，setConfig 以及 parseConfigByNode。
7. 在构造函数中调用父类的 parseResourceByNode，其会调用子类重新的 parseConfigByNode。其初始化的调用堆栈如下。

```
UclampLimitResource::parseConfigByNode
Resource::parseResourceByNode
UclampLimitResource::UclampLimitResource
ResourceManager::parseSysCallResourceByNode
ResourceManager::parseResourceConfigNode
ResourceManager::initResourceConfig
ResourceManager::loadResourceFromConfig
ResourceManager::ResourceManager
ResourceManager::getInstance
```

5.2.3 新增 binder_call 类型 Resource

实现步骤与 syscall 是基本一致的，差异点在于如何获取 binder 服务，以控制 dramdfs 服务为例，描述如何增加 binder 接口。

1. 引入 package，在 Android.bp 中，增加对应的 package。

```
    ],
    shared_libs: [
        "libutils",
        "libcutils",
        "liblog",
        "libbase",
        "libbinder_ndk",
        "android.hardware.power-V3-ndk",
        "vendor.hardware.power.aw-V1-ndk",
        "com.softwinner.dramdfs-V1-ndk",
    ],
    include_dirs: [
        "hardware/libhardware/include"
```

图 5-7: 引入 dramdfs 包

2. 实现控制类。

这里我们不是直接在 Resource 中直接获取对应的 binder 对象的，而是最好新建一个 Controller 来作为连接对象，因为 dramfs 中可能不止一个资源，可能会有多个 resource 需要控制他，故这个 Controller 最好也是单例模式的，实现如下。

hardware/aw/power/aidl/impl/bsp/include/DramDfsController.h
hardware/aw/power/aidl/impl/bsp/DramDfsController.cpp

代码实现。

```
#include <string>
#include <aidl/com/softwinner/dramdfs/IDramDfsService.h>

namespace aidl::android::hardware::power::impl::aw::v2 {

class DramDfsController {
  public:
    static DramDfsController* getInstance();

    void setDfsGovernor(std::string governor);
    void limitNsiBandwidth(bool limit);

  private:
    bool mEnableDramDfsController;

    void init () ;
    bool checkConnect();

    std:: shared_ptr<aidl::com::softwinner::dramdfs::IDramDfsService> mDramDfsService;

    // single pattern
    DramDfsController(void);
    DramDfsController(DramDfsController const&) = delete;
    DramDfsController& operator=(DramDfsController const&) = delete;

    static DramDfsController* instance;

}; //class DramDfsController
```

```
} // namespace aidl::android::hardware::power::impl::aw::v2
```

这里是实现了一个单例模式的 DramDfsController，用于有多个 Resource 时可只持有一份 DramDfsController 即可。

checkConnect 实现

```
bool DramDfsController::checkConnect() {
    if (mDramDfsService != nullptr)
        return true;

    ALOGD("DramDfsController %s", kInstance.c_str());

    :: ndk::SpAIBinder vibBinder = ::ndk::SpAIBinder(AServiceManager_checkService(kInstance.c_str()));
    if (vibBinder.get() != nullptr) {
        mDramDfsService = aidl::com::softwinner::dramdfs::IDramDfsService::fromBinder(vibBinder);
        if (mDramDfsService == nullptr) {
            ALOGW("DramDfsController get DramDfsService fail");
            return false;
        } else {
            return true;
        }
    }
    ALOGW("DramDfsController get DramDfsService binder fail");
    return false;
}
```

这里采用的是 AServiceManager_checkService，而不是使用 wait 或者 get 的方式，原因是因为 wait 和 get 的方式都会等待服务启动，如果服务没启动，可能会 block 一段时间，而 check 是不需要等待的，可以立即返回。

setDfsGovernor 实现

```
void DramDfsController::setDfsGovernor(std::string governor) {
    if (!mEnableDramDfsController)
        return;

    if (checkConnect()) {
        ALOGD("DramDfsController setDfsGovernor %s", governor.c_str());
        mDramDfsService->setDfsGovernor(governor);
    }

}
```

这里可以看到，每次调用接口前，都先调用一下 checkConnect 判断一下是否 mDramDfsService 为空了，为空则不调用了，避免出现异常。

3. 实现 Resource

实现了 Controller 之后，再去实现 Resource，就跟前面的 syscall 没有区别了，按照前面的步骤适配即可，在 setConfig 中调用 Controller 的接口即可。

5.2.4 新增 sysdir 类型 Resource

新增一个 sysdir 形式的 Resource，只需要直接添加到对应的 Resource 文件以及相关的 Scene-Resource 即可。

以温控策略为例。

```
<resource name="thermal_policy" type="sysdir" default="power_allocator">
    <config name="power_allocator"  priority="1">
        <write dir="/sys/class/thermal" read_node="type" read_value="cpul_thermal_zone" write_node="policy" value="power_allocator"/>
        <write dir="/sys/class/thermal" read_node="type" read_value="cpub_thermal_zone" write_node="policy" value="power_allocator"/>
    </config>
</resource>
```

图 5-8: sysdir 示例

参数说明如下：

- dir：需要索引的目录。
- read_node：在 dir 下去读取的节点，例如 dir/{x}/read_node。
- read_value：期望匹配的值，如果与读取 read_node 的值一致，则匹配成功。
- write_node：匹配成功后，需要写入的节点。
- value：匹配成功后，需要对 write_node 写入的值。

以图 [sysdir 示例] 为例，启动时，PowerHal 会去如下路径进行遍历。

```
/sys/class/thermal
```

其遍历访问路径为。

```
/sys/class/thermal/{sub_dir}/type
```

如果读取到 type 为 cpub_thermal_zone，则说明匹配成功，则会自动组成下列路径并记录。

```
/sys/class/thermal/{sub_dir}/policy
```

启动完成后，也可通过 debug 命令来确认 sysdir resource 的初始化情况。

以 thermal_policy 可读取到 resource 的路径参考如下。

```
ResourceId = 0x9 ResoureceName: thermal_policy
  mResourceType=sysdir mDefaultConfig=power_allocator
  mCurrentConfig=power_allocator

  Config1 ConfigName=power_allocator priority=1
    Write1<Dir=/sys/class/thermal ReadNode=type ReadValue=cpul_thermal_zone
          WritePath=/sys/class/thermal/thermal_zone0/policy Value=power_allocator>
    Write2<Dir=/sys/class/thermal ReadNode=type ReadValue=cpub_thermal_zone
          WritePath=/sys/class/thermal/thermal_zone2/policy Value=power_allocator>

  mCurrentItem = NULL
  All  Items :
```

说明

sysdir 与 sysfs 的差异在于 sysdir 需要在启动时自动匹配需要写入的节点路径，当 sysdir 初始化完成后，其调用和 sysfs resource 是没有区别的。

6 使用 Powerhal 进行场景管理

PowerHal 本身并不做相关场景的逻辑判断，其主要作用是一个场景管理的集中入口和系统资源的调度使用；做到一个承上启下的作用。

当某模块需要使用 PowerHal 进行场景管理时，可按照如下步骤进行适配。

1. 明确需要增加的 scene 和对应要控制的 resource 配置；
2. 参考 **DEBUG** 章节修改 xml，并通过相关的 dumpsys 命令调试 OK；如果是特殊类型资源需要开发，可先向 PowerHal 维护人员提出需求；
3. 参考下列相关使用 demo，在模块中完成对应代码的编写；
4. 调试，在关闭 selinux 情况下验证 OK；
5. 完成 selinux 适配，并整理所有补丁提交；
6. 请 PowerHal 维护人员 review 补丁，确认无误后合入补丁。

具体相关 demo 使用示例，可参考下列章节。

6.1 SystemServer 内调用实现

6.1.1 原生场景-Powerhal 使用

原生的 PowerHal 提供了 setMode 和 setBoost 2 个接口，并提供了对应的 Mode 和 Boost 作为原生场景使用。

部分 Mode 和 Boost 已经在谷歌提供的源码中给予实现和调用。对应 Mode 和 Scene 的在 framework 中的定义路径如下。

```
frameworks/base/core/java/android/os/PowerManagerInternal.java
```

以MODE_DISPLAY_INACTIVEMode 为例，该 Mode 表示当所有 display 均关闭（即灭屏）或其中一个打开（亮屏），进入和退出这个场景；其在 framework 中的调用如下。

代码路径。

```
frameworks/base/services/core/java/com/android/server/power/PowerManagerService.java
```

调用实例。

```
import static android.os.PowerManagerInternal.MODE_DISPLAY_INACTIVE;

@Override
public void onDisplayStateChange(boolean allInactive, boolean allOff) {
    // This method is only needed to support legacy display blanking behavior
    // where the display's power state is coupled to suspend or to the power HAL.
    // The order of operations matters here.
    synchronized (mLock) {
        setPowerModeInternal(MODE_DISPLAY_INACTIVE, allInactive);
        if (allOff) {
            if (!mDecoupleHalInteractiveModeFromDisplayConfig) {
                setHalInteractiveModeLocked(false);
            }
            if (!mDecoupleHalAutoSuspendModeFromDisplayConfig) {
                setHalAutoSuspendModeLocked(true);
            }
        } else {
            if (!mDecoupleHalAutoSuspendModeFromDisplayConfig) {
                setHalAutoSuspendModeLocked(false);
            }
            if (!mDecoupleHalInteractiveModeFromDisplayConfig) {
                setHalInteractiveModeLocked(true);
            }
        }
    }
}
```

如原厂希望在进入该模式时，将 TP 设置为低功耗模式以节省功耗，并在 TP 中增加了一个节点开关该模式；增加对应的 scene 和 resource 配置如下。

增加 resource

```
<resource name="tp_idle" type="sysfs" default="wake">
    <config name="wake" priority="1">
        <write path="/sys/class/ctp/tp_idle" value="0"/>
    </config>
    <config name="idle" priority="2">
        <write path="/sys/class/ctp/tp_idle" value="1"/>
    </config>
</resource>
```

增加一个 tp_idle 的 resource，提供 2 种配置，默认为 wake，且 wake 优先级高于 idle。

增加 scene

```
<scene name="DISPLAY_INACTIVE">
    <set resource="tp_idle" value="idle"/>
</scene>
```

增加一个DISPLAY_INACTIVE的 scene，并在该场景时，将 tp_idle 的 resource 设置为 idle。

⚠注意

原生的 Mode 和 Boost 在 scene 配置中，name 必须与 aidl 中的定义完全一致，否则会调用失败。
相关 aidl 的路径如下:
hardware/interfaces/power/aidl/android/hardware/power/Mode.aidl
hardware/interfaces/power/aidl/android/hardware/power/Boost.aidl

6.1.2 非原生场景-AwExtensionPower 使用

6.1.2.1 实现

当前 framework 层实现了一个 AwPowerService，可直接通过 service 调用拓展接口。其源码实现路径

```
frameworks/base/services/core/java/com/aw/server/AwPowerService.java
```

相关的接口说明如下。

setExtensionBoost

- 功能：设置对应的 scene，持续对应的时间。
- 函数原型：public void setExtensionBoost(String sceneName, int durationMs)
- 返回值：void
- 参数
  - **sceneName**：对应 sceneId 的 boost 名字；
  - **durationMs**：使能 scene 的时间，单位为 ms，如果不确定结束时间，则设置为 0，需要提前结束，可设置为-1。

getReadValue

- 功能：获取对应 resource 中可读取的信息，返回值为 Map。
- 函数原型：public Map<String, String> getReadValue(String resourceName)
- 返回值：Map
- 参数
  - **resourceName**：对应需要获取 read 的 resource。

6.1.2.2 使用实例

1. 引入 package。

```
import com.aw.server.AwPowerService
```

2. 获取实例对象。

```
AwPowerService mAwPowerService
```

3. 从 servicemanager 中获取实例化对象。

```
mAwPowerService = LocalServices.getService(AwPowerService.class);
```

4. 调用实例。

```
mAwPowerService.setExtensionBoost(AwPowerService.ENABLE_UCLAMP_LIMIT, -1);
```

6.2 内核使用 uevent 传递 scene

在一些特殊场景下，比如某个场景，可能有多个入口，但最终都会调用到底层统一处理，所以需要在内核来发起场景管理，而内核是无法直接调用 Android 层的，这里采用 uevent 的方式来实现该调用。

即：内核模块通过发送 uevent，PowerHal 接收 uevent 后进行场景的设置，下面是使用例子。

6.2.1 内核模块上报指定 uevent

6.2.1.1 uevent 格式

内核上报给 PowerHal 解析的 uevent，当前采用如下格式约定。

```
SYSTEM=DEMO
EVENT=
TIMEOUT_SEC=
```

说明如下。

- SYSTEM：说明调用系统，用于 PowerHal 匹配对应的场景。
- EVENT：开启情况，当前可定义为：BOOSTREQ、ON、OFF，其中 BOOSTREQ 表示持续一段时间。
- TIMEOUT_SEC：只有当 EVENT 为 BOOSTREQ 使用，定义为该场景开启的时间，单位为秒。

6.2.1.2 上报 uevent

在内核模块中，增加上报 uevent 的函数

```
static int demo_handler(struct demo_device *demo, bool en)
{
        struct device *dev = demo->demo_dev.dev;
        char *envp[3] = {
                "SYSTEM=DEMO",
                NULL,
                NULL};

        if (en)
                envp[1] = "EVENT=ON";
        else
                envp[1] = "EVENT=OFF";

        kobject_uevent_env(&dev->kobj, KOBJ_CHANGE, envp);
        printk("demo is %s\n", en ? "on" : "off");
        return 0;
}
```

在上述代码中，我们从 demo_device 中获取了 device 变量，然后创建了一个 SYSTEM 为 DEMO 的 uevent，并根据 en 的值，来给 EVENT 赋值 ON 或者 OFF；

最后通过 kobject_uevent_env 函数来上报一个 uevent。

6.2.2 PowerHal UeventMonitor 处理 uevent

下列说明中的文件均在 PowerHal 源码中

1. 在 UeventMonitor.h 头文件中，增加对应的 UeventSystem 的枚举。

```
typedef enum {
    UNKNOWN_SYSTEM,
    WIFI,
    CAMERA,
    USB_AUDIO,
    HDMI_DISP,
    DEMO,
} UeventSystem;
```

2. 在 UeventMonitor::processUevent 中，增加对 Uevent 的处理。

```
    if (recvUevent.system && recvUevent.event) {
        if (!strcmp(recvUevent.system, "WIFI")) {
            recvUevent.ueventSystem = UeventSystem::WIFI;
        } else if (!strcmp(recvUevent.system, "CAMERA")) {
            recvUevent.ueventSystem = UeventSystem::CAMERA;
        } else if (!strcmp(recvUevent.system, "USB_AUDIO")) {
            recvUevent.ueventSystem = UeventSystem::USB_AUDIO;
        } else if (!strcmp(recvUevent.system, "HDMI_DISP")) {
            recvUevent.ueventSystem = UeventSystem::HDMI_DISP;
        } else if (!strcmp(recvUevent.system, "DEMO")) {
            recvUevent.ueventSystem = UeventSystem::DEMO;
        }

```

图 6-1: uevent 处理

3. 在 UeventMonitor::onUevent，通过枚举设置对应 scene。

```
void UeventMonitor::onUevent(struct Uevent uevent) {
    std::string sceneName;

    ALOGI("%s request uevent %s durationMs %d", uevent.system, uevent.event, uevent.boostDurationMs);
    switch (uevent.ueventSystem) {
        case WIFI:
            sceneName = std::string("wifi_perf");
            break;
        case CAMERA:
            sceneName = std::string("camera_stream");
            break;
        case USB_AUDIO:
            sceneName = std::string("type_c_audio_output");
            break;
        case HDMI_DISP:
            sceneName = std::string("double_display_output");
            break;
        case DEMO:
            sceneName = std::string("demo_scene");
            break;
        default:
            ALOGD("unknown request system %s event %s", uevent.system, uevent.event);
            return;
    }

    mSceneManager->boost(sceneName, uevent.boostDurationMs);
}
```

图 6-2: 通过 uevent 设置场景

4. 在对应的场景管理配置文件中，增加一个 demo_scene 的 scene，然后设置不同的 resource。

6.3 跨进程调用 AwExtensionPower

当其他非 system_server 需要调用 PowerHal 进行场景管理，可通过 binder 调用的方式进行。

6.3.1 AwExtensionPower 接口说明

AwExtensionPower 提供 2 个接口，接口说明如下。

setExtensionBoost

- 功能：设置对应的 scene，持续对应的时间。
- 函数原型：public void setExtensionBoost(String sceneName, int durationMs)
- 返回值：void
- 参数
  - **sceneName**：对应 sceneId 的 boost 名字；
  - **durationMs**：使能 scene 的时间，单位为 ms，如果不确定结束时间，则设置为 0，需要提前结束，可设置为-1。

getResourceValue

- 功能：获取对应 resource 中可读取的信息，返回值为 Map。
- 函数原型：ndk::ScopedAStatus getResourceValue(const std::string& resourceName, std::vector<ReadEvent>* _aidl_return)
- 返回值：ReadEvent 数组，ReadEvent 为自定义结构体，具体释义见下文。
- 参数
  - **resourceName**：对应需要获取 read 的 resource。

6.3.1.1 ReadEvent

ReadEvent 为自定义的结构体，目的是为了传输 getResourceValue 的返回值，其定义如下。

```
package vendor.hardware.power.aw;

@VintfStability
parcelable ReadEvent {
    String readName;
    String readValue;
}
```

- readName：resource 中 read 类型中对应的 name；
- readValue：该 read 获取的值，如果读取失败，则为空。

6.3.2 Demo 示例

demo 源码路径。

```
hardware/aw/power/aidl/demo
```

可根据需求进行参考。

```
.
├── java
│   ├── Android.bap
│   └── Demo.java
└── ndk
    ├── Android.bap
    ├── demo.c
    ├── Demo.cpp
    └── include
        ├── demo.h
        └── Demo.h
```

技巧

调试过程中，可能会遇到 selinux 报错，可以通过临时关闭 selinux 的方法进行调试，确认接口调试完毕后再来适配 selinux。

6.3.3 Java 端

1. Android.bp 引入包。

```
static_libs : [
    "android.hardware.power-V3-java",
    "vendor.hardware.power.aw-V1-java",
],
```

说明

这里要同时引入android.hardware.power和vendor.hardware.power.aw包。
其中 Android13 里android.hardware.power使用 V3 版本，Android 14 应该使用 V4 版本。

2. 导入 package，并创建对象。

```
import android.hardware.power.IPower;
import vendor.hardware.power.aw.IAwExtensionPower;

public class Demo {
    private IPower mPower;
    private IAwExtensionPower mAwExtensionPower;
}
```

3. 实例化类对象，并通过 getExtension 获取 AwExtensionPower 接口。

```
public class Demo {
    private boolean checkConnect() {
        if (mPower == null || mAwExtensionPower == null) {
            try {
                mPower = IPower.Stub.asInterface(ServiceManager.waitForDeclaredService("android.hardware.power.
```

```
    IPower/default"));
                mAwExtensionPower = IAwExtensionPower.Stub.asInterface(mPower.asBinder().getExtension());
                if (mAwExtensionPower == null) {
                    Slog.d(TAG, "Maybe unsupport AwExtensionPower");
                    return false;
                }
            } catch (Exception e) {
                Slog.w(TAG, "Get PowerHal or ExtensionPower fail", e);
                return false;
            }
        }
        return true;
    }
}
```

3. 调用接口，设置场景。

```
public class Demo {
    private String kDemoScene = "demo";

    public void setPower(int durationMs) {
        if (checkConnect()) {
            try {
                mAwExtensionPower.setExtensionBoost(kDemoScene, durationMs);
            } catch (Exception e) {
                Slog.w(TAG, "set boost fail", e);
            }
        }
    }
}
```

说明

每次调用前，都先检查一下是否连接是否为空，避免出现空指针异常。

6.3.4 CPP 端

PowerHal 原生限制不允许 C++ 调用，如用 c++ 或 c 语言编写的独立服务进程需要调用可参考下方 NDK 端的调用。

6.3.5 NDK 端

如果需要在一个 hal 服务里设置对应的场景管理，请参考下列步骤。

1. Android.bp 引入包名。

```
    shared_libs: [
        "libbinder_ndk",
        "android.hardware.power-V3-ndk",
```

```
        "vendor.hardware.power.aw-V1-ndk",
    ],
```

说明

这里要同时引入android.hardware.power和vendor.hardware.power.aw包。
其中 Android13 里android.hardware.power使用 V3 版本，Android 14 应该使用 V4 版本。
同时还需要引入 libbinder_ndk 库。

2. include 头文件，创建类对象。

```
#include <aidl/android/hardware/power/IPower.h>
#include <aidl/vendor/hardware/power/aw/IAwExtensionPower.h>

namespace xxxx {

class Demo {
    bool checkConnect();
    void setPower(int durationMs);

    std:: shared_ptr<aidl::android::hardware::power::IPower> mPower;
    std:: shared_ptr<aidl::vendor::hardware::power::aw::IAwExtensionPower> mAwExtensionPower;
};

} // namespace xxxx
```

3. 实例化类对象，并通过 getExtension 获取 AwExtensionPower 接口。

```
#include <android/binder_manager.h>

namespace xxxx {

static const std:: string kInstance = std:: string() + aidl:: android::hardware::power::IPower::descriptor + "/default";
static const std:: string kDemoString = std::string("demo_scene");

bool Demo::checkConnect() {
    if (mPower != nullptr && mAwExtensionPower != nullptr) {
        return true;
    }

    :: ndk::SpAIBinder vibBinder = ::ndk::SpAIBinder(AServiceManager_checkService(kInstance.c_str()));
    if (vibBinder.get() != nullptr) {
        mPower = aidl::android::hardware::power::IPower::fromBinder(vibBinder);
        if (mPower != nullptr) {
            :: ndk::SpAIBinder cvibBinder;
            AIBinder_getExtension(vibBinder.get(), cvibBinder.getR());
            if (cvibBinder.get() != nullptr) {
                mAwExtensionPower = aidl::vendor::hardware::power::aw::IAwExtensionPower::fromBinder(cvibBinder);
                if (mAwExtensionPower != nullptr) {
                    ALOGE("Demo get mAwExtensionPower success");
                    return true;
                }
            }
        }
    }
```

```
    return false;
}

} // namespace xxxx
```

4. 调用接口，设置场景。

```
namespace xxxx {
static  const std:: string  kDemoString = std::string("demo_scene");

void Demo::setPower(int durationMs) {
    if  (checkConnect()) {
        mAwExtensionPower->setExtensionBoost(kDemoString, durationMs);
    }
}
} // namespace xxxx
```

说明

针对 c 语言编写的代码，也可通过使用 c 的 demo 去调用。

6.3.6 验证

接口通了以后，如何确认场景是否打开。

1. 关闭 selinux；
2. logcat 中查看 AW_PowerHAL，看是否有对应的场景设置信息。

```
logcat |  grep AW_PowerHAL
```

设置参考语句如下：

进入场景，其中 0 表示永久。

```
I  AW_PowerHAL: set scene demo(0x70000009) duration 0
```

设置指定时间场景。

```
I  AW_PowerHAL: set scene demo(0x70000009) duration 5000
```

该打印表示进入 demo 场景 5s。

退出场景打印。

```
I  AW_PowerHAL: set scene demo(0x70000009) duration -1
```

如果是超时场景退出，则分 2 种情况：

1. timeout 时间未到达，提前结束场景；

```
I  AW_PowerHAL: resource: cancel scene(0x70000009) before request item timeout
```

2. timeout 时间到达，正常结束场景。

```
I  AW_PowerHAL: resource: scene(0x70000009) timeout, end it
```

3. 通过 dumpsys 查看场景是否设置成功。

以 demo 场景为例，通过下列命令查看 demo 场景设置的资源。

```
dumpsys android.hardware.power.IPower/default -s
```

所需资源如下。

```
SceneId = 0x70000009 SceneName: demo
  mEnable=1  isSupport=1
  Scene Sets :
    Set1<Resource=dfs_governor Value=performance>
```

表示设置 dfs_governor 为 performance。

设置场景后，通过下列命令查看是否设置成功。

```
dumpsys android.hardware.power.IPower/default -r
```

设置成功的打印。

```
ResourceId = 0x2 ResoureceName: dfs_governor
  mResourceType=binder_call mDefaultConfig=sunxi_actmon
  mCurrentConfig=performance

  Config1 ConfigName=performance priority=1
    Write1<interface=setDfsGovernor targetGov=performance>
  Config2 ConfigName=sunxi_actmon priority=2
    Write1<interface=setDfsGovernor targetGov=sunxi_actmon>

  mCurrentItem = [mCallerId=0 mSceneId=0x70000009 mConfigName=performance mRequestTimeMs=0]
  All  Items :
    Item1 : [mCallerId=0 mSceneId=0x70000009 mConfigName=performance mRequestTimeMs=0]
```

可见已设置成功。

6.3.7 selinux 权限

接口调试通了以后，可以先在对应进程的 selinux 的 te 文件中，增加如下 selinux 语句。

```
hal_client_domain(demo_default, hal_power);
```

这句话的意思是，注册成 hal_power 的 client 端，这样就可以调用 server 端，而 powerhal 是默认注册成 server 端的，故通常加上这句后，一般就 OK 了。

如果还有其他 selinux 报错，可自行解决，或咨询 powerhal 开发人员。

6.3.8 编译报错

6.3.9 找不到 package

如果遇到下列报错。

```
cedarx-config: media afbc mode: [2]
cedarc-config: sdkVersion[33], board[saturn], platformconfig[YES], cryptolevel[1], playreadytype[]
registry_table has NOT been set. Please check it.
registry_table has NOT been set. Please check it.
error: hardware/aw/vibrator/aidl/Android.bp:28:1: "android.aw.hardware.vibrator-service" depends on undefined module "android.hardware.power-V3-ndk"
error: hardware/aw/vibrator/aidl/Android.bp:28:1: "android.aw.hardware.vibrator-service" depends on undefined module "vendor.hardware.power.aw-V1-ndk"
10:36:10 soong bootstrap failed with: exit status 1

###### failed to build some targets (19 seconds) ####
```

图 6-3: 找不到包

请查看包名中间的-是否为中文字符？如果是从文档中拷贝的会存在这种情况。

6.3.10 找不到变量

```
hardware/aw/vibrator/aidl/Vibrator.cpp:44:33: error: unexpected character <U+2011>
static int kFaceUnlockSceneId = -1;
                                ^

hardware/aw/vibrator/aidl/Vibrator.cpp:222:29: error: unexpected character <U+2011>
            if (kFaceUnlockSceneId == -1) {
                                ^

hardware/aw/vibrator/aidl/Vibrator.cpp:223:32: error: character <U+2011> not allowed in an identifier
                    auto ret = mAwExtensionPower->getSceneIdByName(kFaceUnlockString, &kFaceUnlockSceneId);
                                                 ^

hardware/aw/vibrator/aidl/Vibrator.cpp:223:36: error: use of undeclared identifier 'getSceneIdByName'
                    auto ret = mAwExtensionPower->getSceneIdByName(kFaceUnlockString, &kFaceUnlockSceneId);
                                                   ^

hardware/aw/vibrator/aidl/Vibrator.cpp:223:15: error: use of undeclared identifier 'mAwExtensionPower-'
                    auto ret = mAwExtensionPower->getSceneIdByName(kFaceUnlockString, &kFaceUnlockSceneId);
                               ^

hardware/aw/vibrator/aidl/Vibrator.cpp:228:7: error: use of undeclared identifier 'kFaceUnlockSceneIdd'; did you mean 'kFaceUnlockSceneId'?
            if (kFaceUnlockSceneIdd >= 0) {
                ^~~~~~~~~~~~~~~~~~~
                kFaceUnlockSceneId
hardware/aw/vibrator/aidl/Vibrator.cpp:44:12: note: 'kFaceUnlockSceneId' declared here
static int kFaceUnlockSceneId = -1;
           ^

hardware/aw/vibrator/aidl/Vibrator.cpp:229:21: error: character <U+2011> not allowed in an identifier
                    mAwExtensionPower->setExtensionBoost(kFaceUnlockSceneId, kFaceUnlockString, durationMs);
                                      ^

hardware/aw/vibrator/aidl/Vibrator.cpp:229:4: error: use of undeclared identifier 'mAwExtensionPower-'; did you mean 'mAwExtensionPower'?
                    mAwExtensionPower->setExtensionBoost(kFaceUnlockSceneId, kFaceUnlockString, durationMs);
                    ^~~~~~~~~~~~~~~~~~
                    mAwExtensionPower
hardware/aw/vibrator/aidl/include/vibrator-impl/Vibrator.h:64:76: note: 'mAwExtensionPower' declared here
    std:: shared_ptr<aidl::vendor::hardware::power::aw::IAwExtensionPower> mAwExtensionPower;
```

图 6-4: 无法找到变量

这种情况可能是因为从文档中复制代码，会复制成中文字符，相关代码复制使用请参考 demo 路径例子。

7 DEBUG

7.1 dumpsys 命令

7.1.1 参数说明

当前使用原生的 dumpsys 接口实现了 debug 模块，当前支持的 debug 参数如下。

命令如下：

```
dumpsys android.hardware.power.IPower/default
```

其参数说明如下。

```
PowerHal Current version: V2.1.2(20250409-406a23ae06b7)
Usage: program [OPTIONS]
  Options:
    -h, -- help                 Display this  help message
    -a, -- all                  Show all scenes and resources
    -s, -- scene                Show scenes only
    -r, -- resource             Show resources only
    -b, -- boost SCENENAME MS  Set boost for a scene (NAME) with duration in milliseconds (MS)
    -c, -- cat  RESOURCENAME    Cat RESOURCENAME with all reads value
```

通过命令，可以查看当前整个系统支持的场景，以及当前资源的状态，同时还支持通过 dumpsys 设置 scene。

- h：显示帮助，可查看当前 PowerHal 的版本以及编译时间；
- a：显示当前的支持的 scene 和所有 resource 的状态；
- s：仅显示 scene；
- r：仅显示 resource 的状态；
- b：手动设置某个场景，用于 debug，需要 2 个参数，scene 的 name 以及持续时间 time，其中，0 表示常开该场景，-1 表示退出场景，大于 0 表示持续的时间，单位为 ms，每次开启不低于 500ms；
- c：读取某个 resource 中所有 read 的值，需要 1 个参数，resource 的 name。

7.1.2 使用实例

7.1.2.1 获取 scene 信息

使用如下命令获取当前机器使用的 scene 信息。

```
dumpsys android.hardware.power.IPower/default -s
```

分析 scene 信息。

```
All  Scene Config list:
  SceneId = 0x00000001 SceneName: INTERACTIVE
    mEnable=1  isSupport=1
    Scene Sets :
      Set1<Resource=cpu_governor Value=schedutil>
      Set2<Resource=dfs_governor Value=sunxi_actmon>
      Set3<Resource=io_schedule Value=mq_deadline_low_latency>
      Set4<Resource=thermal_control Value=enabled>
```

如上所示，该 scene 的说明如下：

- SceneId：scene 的 id；
- SceneName：scene 名字；
- Sets：当前 scene 需要设置的 resource 集合。

7.1.2.2 获取 resource 信息

使用如下命令获取当前机器使用的 resource 信息。

```
dumpsys android.hardware.power.IPower/default -r
```

分析 resource 信息。

```
ResourceId = 0x1 ResoureceName: cpu_governor
  mResourceType=sysfs mDefaultConfig=schedutil
  mCurrentConfig=schedutil

  All  Reads :
       Read1 <Readname=cpub_governor Path=/sys/devices/system/cpu/cpufreq/policy6/scaling_governor>
  Config1 ConfigName=performance priority=1
    Write1<TargetPath=/sys/devices/system/cpu/cpufreq/policy0/scaling_governor Value=performance>
    Write2<TargetPath=/sys/devices/system/cpu/cpufreq/policy4/scaling_governor Value=performance>
    Write3<TargetPath=/sys/devices/system/cpu/cpufreq/policy6/scaling_governor Value=performance>
  Config2 ConfigName=schedutil priority=2
    Write1<TargetPath=/sys/devices/system/cpu/cpufreq/policy0/scaling_governor Value=schedutil>
    Write2<TargetPath=/sys/devices/system/cpu/cpufreq/policy4/scaling_governor Value=schedutil>
    Write3<TargetPath=/sys/devices/system/cpu/cpufreq/policy6/scaling_governor Value=schedutil>
    Write4<TargetPath=/sys/devices/system/cpu/cpufreq/schedutil/rate_limit_us Value=20000>

  mCurrentItem = [mCallerId=0 mSceneId=0x00000001 mConfigName=schedutil mRequestTimeMs=0]
```

```
All Items:
  Item1: [mCallerId=0 mSceneId=0x00000001 mConfigName=schedutil mRequestTimeMs=0]
```

如上所示，该 resource 的说明如下：

- ResourceId：resource 的 id；
- ResoureceName：resource 的 name，用于在 scene 中设置对应的 resource 时使用
- mResourceType：resource 的 type；
- mDefaultConfig：当无请求时，该 resource 的默认配置；
- mCurrentConfig：resource 当前使用的配置；
- All Reads：该 resource 中支持读取的 read 配置，这里为支持读取的 cpub 的当前 governor；
- Config：resource 可支持的配置合集，priority 为优先级，数字越小优先级越高
- mCurrentItem：记录当前设置了该 resource 的 scene，这里表示由 sceneid 为 0x00000001 的 scene 设置了 schedutil 的 config，请求时间 0 为常开；
- All Items：记录当前所有设置了该 resource 的 scene。

7.1.2.3 手动 boost 场景

使用如下命令可手动进入某一场景。

```
dumpsys android.hardware.power.IPower/default -b SceneName time(ms)
```

使用说明如下：

- SceneName：前面获取 scene 信息中的 SceneName；
- time(ms)：设置的时间，其中 0 为常开，-1 退出，大于 0 为持续时间。

设置完成后，可通过dumpsys android.hardware.power.IPower/default -r来确认场景是否手动进入和退出。

7.1.2.4 读取 resource 中 read 值

使用如下命令可手动读取 resource 中的 read 值。

```
dumpsys android.hardware.power.IPower/default -c ResourceName
```

使用说明：

- ResourceName：前面获取 ResourceName 信息中的 read 配置的值；

使用步骤

1. 先通过-r 参数，获取到对应的 resource 和相关的 read 信息。

```
dumpsys android.hardware.power.IPower/default -r
```

2. 再通过-c 参数，获取到相关的 resource 的信息。

```
dumpsys android.hardware.power.IPower/default -c cpu_governor
```

示例：

```
Resource(cpu_governor) all reads value list:
readname=cpub_governor value=<schedutil>
```

7.2 remount 后 push 配置文件

在开发过程中，如果只需要修改配置文件，而不修改代码，则可将机器 remout 后，再推入新的配置文件，重启设备来使新的配置文件生效的方法。

使用过程可参考如下：

1. 将机器 remout；
2. 通过 adb push 推入新的配置文件；
3. 重启；
4. 使用 dumpsys 命令查看配置文件是否生效。

所有配置文件在设备中的路径如下：

```
/vendor/etc/aw_power_scene_config.xml
/vendor/etc/aw_power_resource.xml
```

著作权声明

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。