



# HAL接口定义、实现和调用

SLab - LGSI Dev. Group 1

LGSI Charge SME

智磊鑫

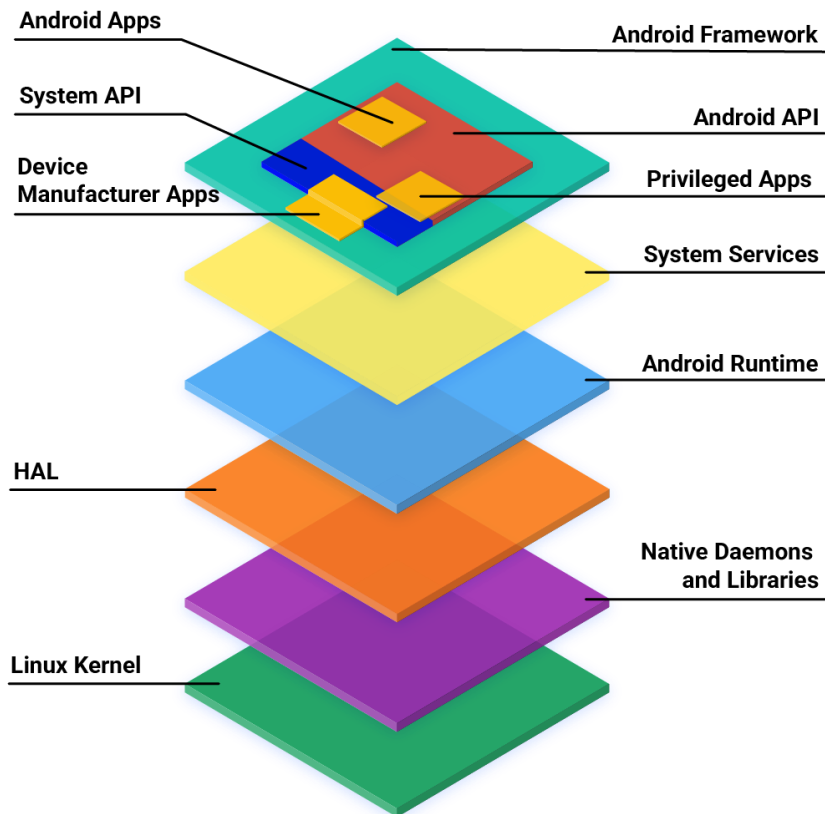
- 
- 01 Android HAL
  - 02 HIDL定义和实现
  - 03 AIDL定义和实现
  - 04 HAL接口的兼容调用
  - 05 Question & Answer



01

# Android HAL<sup>•</sup>

# HAL层概述



HAL (Hardware Abstraction Layer)是一个抽象层，其中包含硬件供应商(Vendor)要实现的标准接口。借助HAL，Android可以忽略较低级别的驱动程序实现，并且我们可以顺利实现相关功能，而不会影响或更改更高级别的系统。

HAL优点：

- 模块化
- 支持HIDL
- 支持AIDL HAL(Android 11之后)
- HIDL HAL可保证Android核心系统（system.img或框架）向后兼容。

# Android Treble

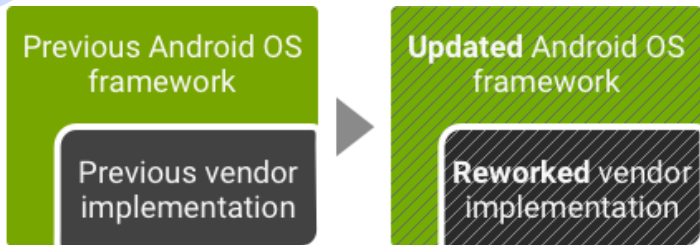


图 1. Treble 推出前的 Android 更新环境

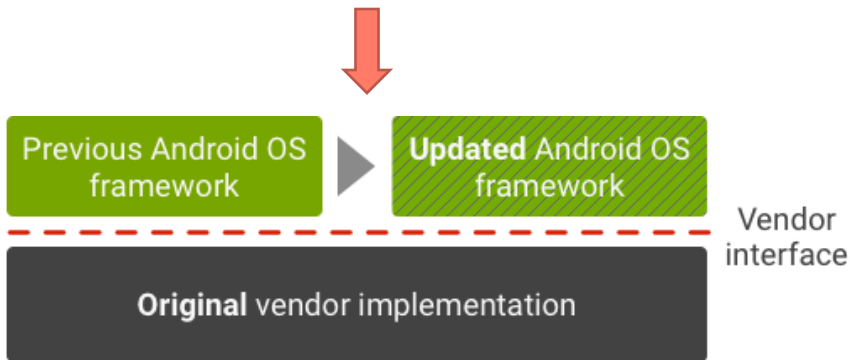


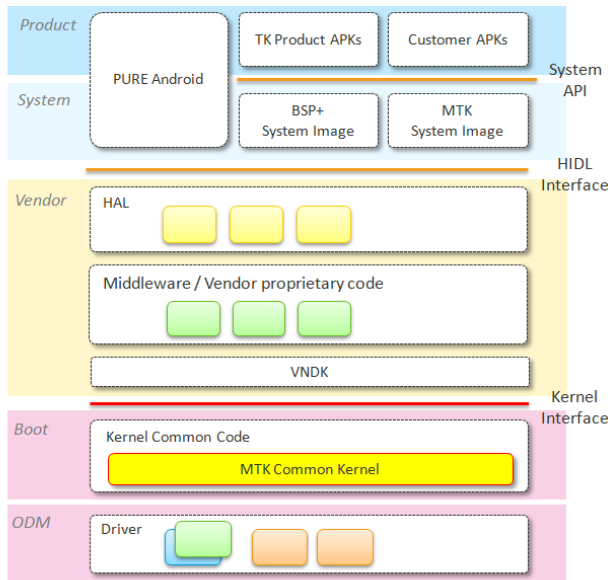
图 2. Treble 推出后的 Android 更新环境

Project Treble 提供了一个稳定的新供应商接口(Vendor interface), 供设备制造商访问 Android 代码中特定于硬件的部分, 这样一来, 设备制造商只需更新 Android 操作系统框架, 即可跳过芯片制造商直接提供新的 Android 版本。

- HAL 类型: 提供了关于绑定式 HAL、直通 HAL、Same-Process (SP) HAL 和旧版 HAL 的说明。
- VNDK(供应商原生开发套件): 提供了关于VNDK (专门用来让供应商实现其HAL的一组库) 的说明
- VINTF(供应商接口对象): VINTF对象整合了关于设备的相关信息, 并让这类信息可以通过可查询API提供。
- SELinux for Android
- ...

# Android 框架

## Android Software Layers



同时从Android Q开始，MTK进行了大规模的设计重构：

1. 将平台无关的代码放入到system Image中，system image中的代码原则上都与平台无关。
2. 使用HAL层进行严格的隔离，严格禁止直接跨system/vendor的操作行为。
3. 将平台相关配置与平台无关配置分离，平台相关配置放在vendor中，平台无关配置则放在system中。
4. 对repo进行审查，针对单个repo同时跨system/vendor的情况进行调整，尽可能地确保一个repo 只属于system或vendor。
5. 达成MSSI，MSSI必须具备动态HAL探测能力，一份MSSI可同时支持多个平台。



02

# HIDL接口定义 和实现

# HIDL实现流程



## 接口概述

HIDL接口的描述、设计和生命周期

## 接口定义

HIDL接口定义、软件包实现、继承、哈希值、设备清单和FCM兼容性矩阵

## 服务实现

HIDL Service的实现和编译规则创建

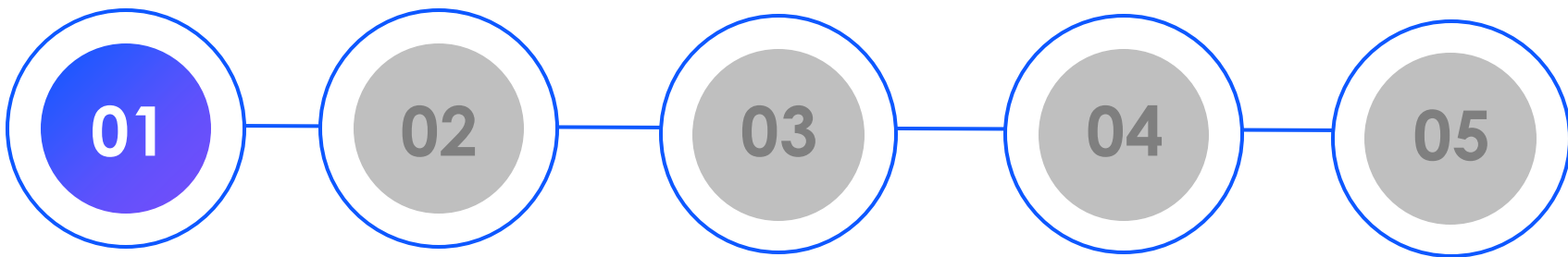
## 注册和发现服务

服务的注册、发现和调用

## 版本控制

版本控制和升级





接口概述

接口定义

服务实现

注册和发现服务

版本控制

# 01 HIDL 概述


1. **HAL** 接口定义语言（简称 **HIDL**）是用于指定 HAL 和其用户之间的接口的一种接口描述语言 (IDL)。
2. **HIDL** 旨在用于进程间通信 (IPC)。
3. **HIDL** 可指定数据结构和方法签名，这些内容会整理归类到接口中，而接口会汇集到软件包中。
4. **HIDL** HAL 可保证 Android 核心系统（也称为 **system.img** 或框架）向后兼容。

# 01 HIDL概述-设计

HIDL的目标:

Android 框架可以在无需重新构建 HAL 的情况下进行替换。HAL 将由供应商或 SOC 制造商构建，并放置在设备的 vendor 分区中，这样一来，就可以在 Android 框架自己的分区中通过 OTA 替换框架，而无需重新编译 HAL。

HIDL 设计在以下方面之间保持了平衡：

- **互操作性**      可以使用各种架构、工具链和 build 配置来编译的进程之间创建可互操作的可靠接口
- **效率**            HIDL 会尝试尽可能减少复制操作的次数
- **直观**      无论是将数据传递到 HIDL 中以进行传输，还是从 HIDL 接收数据，都不会改变数据的所有权

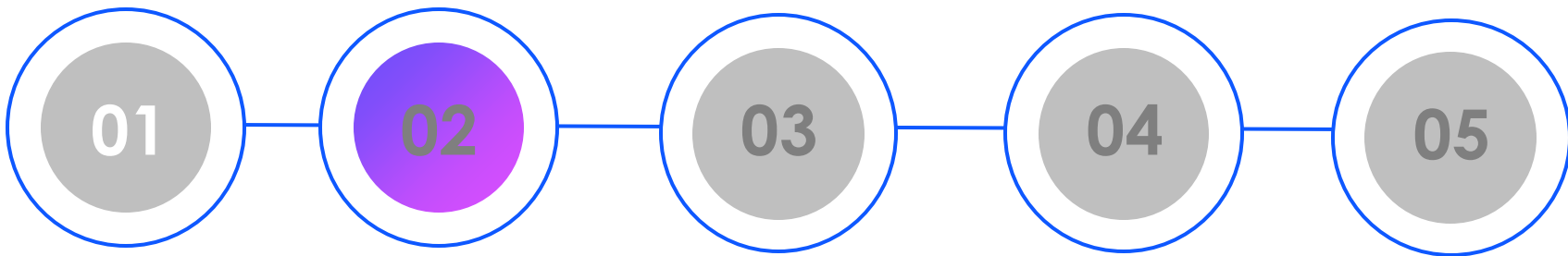
# 01 HIDL概述-生命周期

Android 9 支持在不使用或不需要 Android 硬件子系统时动态关停这些子系统。

在 Android 9 中，必须手动确定 HAL 退出。对于 Android 10 及更高版本，还可以使用自动生命周期确定 HAL 退出。

动态关停：使用 LazyServiceRegistrar 而不是成员函数 registerAsService 通过 C++ 注册服务

```
# some init.rc script associated with the HAL
service vendor.some-service-name /vendor/bin/hw/some-binary-service
    # init language extension, provides information of what service is served
    # if multiple interfaces are served, they can be specified one on each line
    interface android.hardware.light@2.0::ILight default
    # restarted if hwservice manager dies
    # would also cause the hal to start early during boot if disabled wasn't set
    class hal
    # will not be restarted if it exits until it is requested to be restarted
    oneshot
    # will only be started when requested
    disabled
    # ... other properties
```



接口概述

接口定义

服务实现

注册和发现服务

版本控制

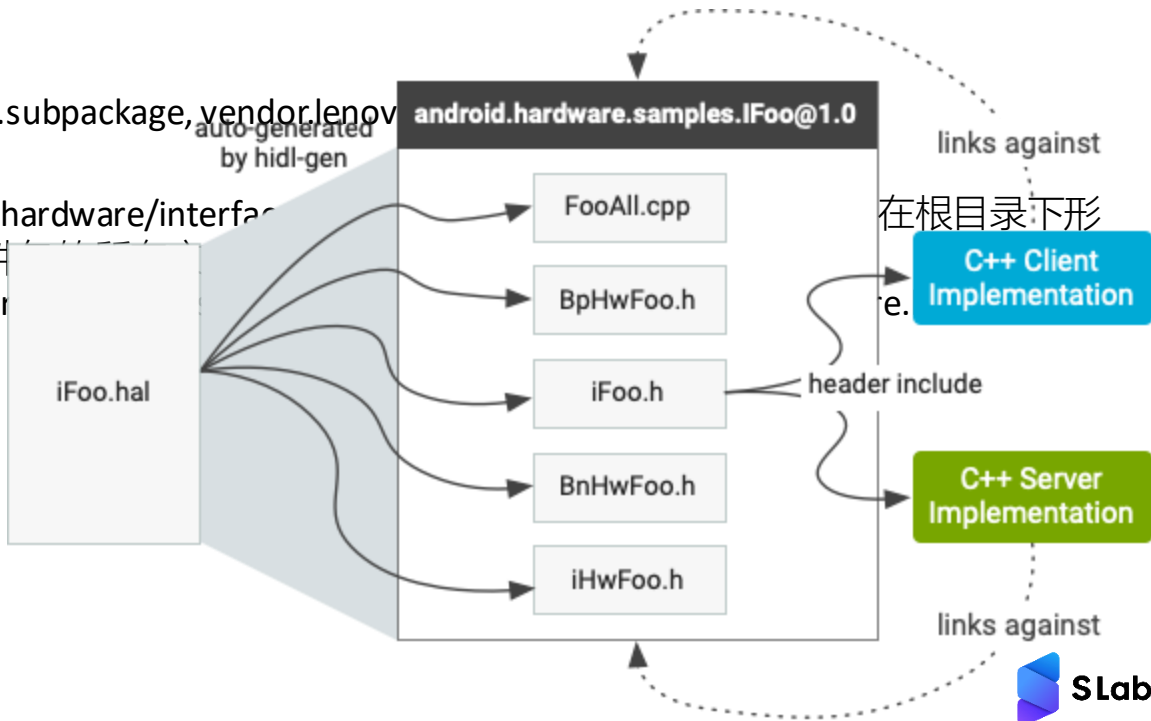
## 02 接口定义-接口和软件包

HIDL代码结构：用户已定义的类型、接口和软件包

HIDL 围绕接口构建而成，而接口是在面向对象的语言中用来定义行为的抽象类型。每个接口都是软件包的一部分。

软件包名称可以具有子级，如 `package.subpackage.vendor.lenovo`

对于已发布的 HIDL 软件包的根目录为 `hardware/interfaces`，可以包含一个或多个子目录；定义同一个软件包的接口，例如，我们可以在 `lenovo/lsi/opensource`



## 02 接口定义-接口和软件包

软件包目录中包含扩展名为 `.hal` 的文件。每个文件均必须包含一个指定文件所属的软件包和版本的 `package` 语句。

`types.hal` 文件并不定义接口，而是定义软件包中每个接口可以访问的数据类型。

软件包前缀	位置	接口类型
<code>android.hardware.*</code>	<code>hardware/interfaces/*</code>	HAL
<code>android.frameworks.*</code>	<code>frameworks/hardware/interfaces/*</code>	frameworks/ 相关
<code>android.system.*</code>	<code>system/hardware/interfaces/*</code>	system/ 相关
<code>android.hidl.*</code>	<code>system/libhidl/transport/*</code>	核心
<code>vendor.lenovo.hardware.*</code>	<code>lenovo/lgsi/opensource/interfaces/*</code>	lenovo HAL

```
zhilx1@byws013:~/Downloads/android_source$ tree .
.
├── 1.0
│   ├── Android.bp
│   └── IBattery.hal
├── 2.0
│   ├── Android.bp
│   └── IBattery.hal
└── package_vendor.lenovo.hardware.battery@1.0;

package_vendor.lenovo.hardware.battery@1.0;
/**
 * 0表示正常，1表示PD_PPS，2表示FLOAT类型
 */
getChargerType() generates(int32_t type);

/**
 * 该方法用来判断是否disable usb供电
 * 当setUsbSupplyDisabled(1)时，pmic usbin 高阻模式，不为系统供电
 * 当setUsbSupplyDisabled(0)时，usb恢复系统供电
 */
setUsbSupplyDisabled(bool enable) generates(bool success);

/**
 * 该方法用来判断是否disable 充电
 * 当setBatteryChargeDisabled(1)时，充电disable
 * 当setBatteryChargeDisabled(0)时，充电enable，根据充电功能，该后动快充的功能
 */
setBatteryChargeDisabled(bool enable) generates(bool success);

setBatteryMaintenanceEnabled(bool enable) generates(bool success);
isBatteryMaintenanceEnabled() generates(bool success);
```

## 02 接口定义-继承

接口可以是之前定义的接口的扩展：

- 接口/软件包 可以向其他 接口/软件包 添加功能，并按原样纳入其 API。
- 接口可以从软件包或特定接口导入类型。
- 接口只能扩展一个其他接口（不支持多重继承）。
- 具有非零 Minor 版本号的软件包中的每个接口必须扩展一个以前版本的软件包中的接口。
- 接口扩展并不意味着生成的代码中存在代码库依赖关系或跨 HAL 包含关系，接口扩展只是在 HIDL 级别导入数据结构和方法定义。HAL 中的每个方法必须在相应 HAL 中实现。

```
1 package vendor.lenovo.hardware.battery@2.0;
2
3 import vendor.lenovo.hardware.battery@1.0::IBattery;
4
5 interface IBattery extends @1.0::IBattery{
6     /**
7      * enable&disable battery maintenance V2
8      * @return true if success, false if failed
9      */
10    setBatteryMaintenanceEnabledV2(bool enable) generates(bool success);
11
12    /**
13     * get battery maintenance V2
14     */
15    getBatteryMaintenanceEnabledV2() generates(bool success);
16 };
```



## 02 接口定义-哈希值

HIDL 接口哈希，这是一种旨在防止接口遭到意外更改并确保接口更改经过全面审查的机制。这种机制是必需的，因为 HIDL 接口带有版本编号，也就是说，接口一经发布便不得再更改，但不会影响应用二进制接口 (ABI) 的情况（例如更正备注）除外。

每个软件包根目录（即映射到 hardware/interfaces 的 android.hardware 或映射到 vendor/lenovo/hardware/interfaces 的 vendor.lenovo.xxx）都必须包含一个列出所有已发布 HIDL 接口文件的 current.txt 文件。

```
zhilxl@byws013:~/Downloads/android_source/lgsi-t-15.0/lenovo/lgsi/opensource/interfaces$ cat current.txt
aff8944024ef80a3f39c1774d9046e4e1fb871dc26ad1ee5841534ee1ebcc3d9 vendor.lenovo.hardware.factory@1.0::types
be55c584a5c01957ea9d7a9bc32e42022ceel1a117b283d4e2a7db72a2485b0d9 vendor.lenovo.hardware.factory@1.0::IFactory
37cb22704eed46811cc66ed0bfc87b97ca7ae35e6b926ba57c15ba30d285ca9c vendor.lenovo.hardware.touchscreen@1.0::types
92db326881147cd0c75a79132beb3418ad04bcf26b4e56a462f6e4640da4ba54 vendor.lenovo.hardware.touchscreen@1.0::ITouchscreen
76928d606ecaebf795a4b1520aa0cd96f0260c58ce67e3fab47a4fba97f301b6 vendor.lenovo.hardware.misc@1.0::types
1ab3744e9a06a8e0e955a060210cc7603bf1a4bf1f7ca9629d685cb9c9cc73188 vendor.lenovo.hardware.misc@1.0::IMisc
b18f0627e975a6055dba516213fc9ee77f44d100065b075b84e0b7e00d733958 vendor.lenovo.hardware.keyboard@1.0::IKeyboard
d00681c7c0baa54ef408758a6d9a54d60ad9233456affa1bc4e06591618eeb46 vendor.lenovo.hardware.performance@1.0::IPerformance
c07d7401fb67acdfa6e244fdf829555574fb2b4022f8387bb53e6f65e17ee9b6 vendor.lenovo.hardware.display@1.0::IDisplay
4ea70d1454c58e4312df8b8618676d268ee468491fa82e4f7affa1c012fac9ab vendor.lenovo.hardware.display@1.0::IDisplayEffect
604b7df6b1d42f51558b9bf6dfafcc23a9b2feb4f58921ccalbb13913d3340383 vendor.lenovo.hardware.battery@1.0::IBattery
92ba65ff5ed84af03254259bbe9a260e645bbfa17519734336c3d43eccb53922 vendor.lenovo.hardware.battery@2.0::IBattery
8368cd93472b343146b86813005f36f50ff8266a72bee6641a8b3a7d59b61108 vendor.lenovo.hardware.ifaa@2.0::types
d09a14d9408b9c34844d4c427e4fd00549248f194986e4a6d9e25d7b6cf75d2b vendor.lenovo.hardware.ifaa@2.0::IIFAADevice
1cb2187c4165ac4c912618a40046c75210bee889eeb85bfe65c6705f6cc0bbc0 vendor.lenovo.hardware.soter@1.0::types
c54977eba2d5c3c17634f94aaadebe7ed9a03e605864ecb71d5a3fae0afd4f26 vendor.lenovo.hardware.soter@1.0::ISoter
1f85707fef7530b801e2d4b5d56bb579f6916d31be3db3d3ee20374086f1a0f2 vendor.lenovo.hardware.virtualinput@1.0::IVirtualInput
1be76482a386ceea908c6ea8bb3390d789f53c6f041fe60569dcd361535e7c1b motorola.hardware.input@1.0::types
d380b6b3f1c124fd83ddefaec04c684a8b85763470192e7310d362e92fe8e11 motorola.hardware.input@1.0::IClientToken
c31e4e0f790ba0c1baff6c0fbb542c2e7338b90617ee5c07c12f94747d8aeece8 motorola.hardware.input@1.0::IMotInput
61a137787cfb1dd0ccb6d21285b9a86d2ecf207a58134036dcaeeafdaf258a0ba motorola.hardware.input@1.1::types
c499867c22fe50689b314a7d827cb92bac256c031c1304459c9b08cf1337d5ab motorola.hardware.input@1.1::IMotInput
```

## 02 接口定义-哈希值

我们可以手动将哈希添加到 current.txt 文件中，也可以使用 hidl-gen 添加。以下代码段提供了可与 hidl-gen 搭配使用来管理 current.txt 文件：

```
hidl-gen -L hash -r vendor.lenovo.hardware.battery:lenovo/lgsi/opensource/interfaces/battery  
vendor.lenovo.hardware.battery@1.0
```

hidl-gen 生成的每个接口定义库都包含哈希，并可通过调用 IBase::getHashChain 进行检索。hidl-gen 编译接口时，会检查 HAL 软件包根目录中的 current.txt 文件，查看 HAL 是否已更改：

- 如果没有找到 HAL 的哈希，则接口会被视为未发布（处于开发阶段），并且编译会继续进行。
- 如果找到了相应哈希，则会对照当前接口对其进行检查：
- 如果接口与哈希匹配，则编译会继续进行。
- 如果接口与哈希不匹配，则编译会暂停，因为这意味着之前发布的接口会被更改。

其他所有更改都应在接口的次要或主要版本升级中进行。

## 02 接口定义-哈希值

```
zhilx1@byws013:~/Downloads/android_source/asphalt/vendor/LINUX/android$ hidl-gen
Usage: hidl-gen -o <output path> -L <language> [-O <owner>] [ -p <root path> ] (-r <interface root>)+ [-R] [-F] [-v] [-d <depfile>] FQNAME...

Process FQNAME, PACKAGE(.SUBPACKAGE)*@[0-9]+.[0-9]+(:.TYPE)?, to create output.

-h: Prints this menu.
-L <language>: The following options are available:
  check      : Parses the interface to see if valid but doesn't write any files.
  c++        : (internal) (deprecated) Generates C++ interface files for talking to HIDL interfaces.
  c++-headers : (internal) Generates C++ headers for interface files for talking to HIDL interfaces.
  c++-sources : (internal) Generates C++ sources for interface files for talking to HIDL interfaces.
  export-header : Generates a header file from @export enumerations to help maintain legacy code.
  c++-impl    : Generates boilerplate implementation of a hidl interface in C++ (for convenience).
  c++-impl-headers: c++-impl but headers only.
  c++-impl-sources: c++-impl but sources only.
  c++-adapter : Takes a x.(y+n) interface and mocks an x.y interface.
  c++-adapter-headers: c++-adapter but helper headers only.
  c++-adapter-sources: c++-adapter but helper sources only.
  c++-adapter-main: c++-adapter but the adapter binary source only.
  java       : (internal) Generates Java library for talking to HIDL interfaces in Java.
  java-impl  : Generates boilerplate implementation of a hidl interface in Java (for convenience).
  java-constants : (internal) Like export-header but for Java (always created by -Lmakefile if @export exists).
  vts        : (internal) Generates vts proto files for use in vtsd.
  makefile   : (removed) Used to generate makefiles for -Ljava and -Ljava-constants.
  androidbp  : (internal) Generates Soong bp files for -Lc++-headers, -Lc++-sources, -Ljava, -Ljava-constants, and -Lc++-adapter.
  androidbp-impl : Generates boilerplate bp files for implementation created with -Lc++-impl.
  hash       : Prints hashes of interface in 'current.txt' format to standard out.
  function-count : Prints the total number of functions added by the package or interface.
  dependencies : Prints all depended types.
  inheritance-hierarchy: Prints the hierarchy of inherited types as a JSON object.
  format      : Reformats the .hal files
-O <owner>: The owner of the module for -Landroidbp(-impl)?
-o <output path>: Location to output files.
-p <root path>: Android build root, defaults to $ANDROID_BUILD_TOP or pwd.
-R: Do not add default package roots if not specified in -r.
-F: Require all referenced ASTs to be frozen.
-r <package:path root>: E.g., android.hardware:hardware/interfaces.
-v: verbose output.
-d <depfile>: location of depfile to write to.
```

生成对应的文件，如Android.bp，.h .cpp等

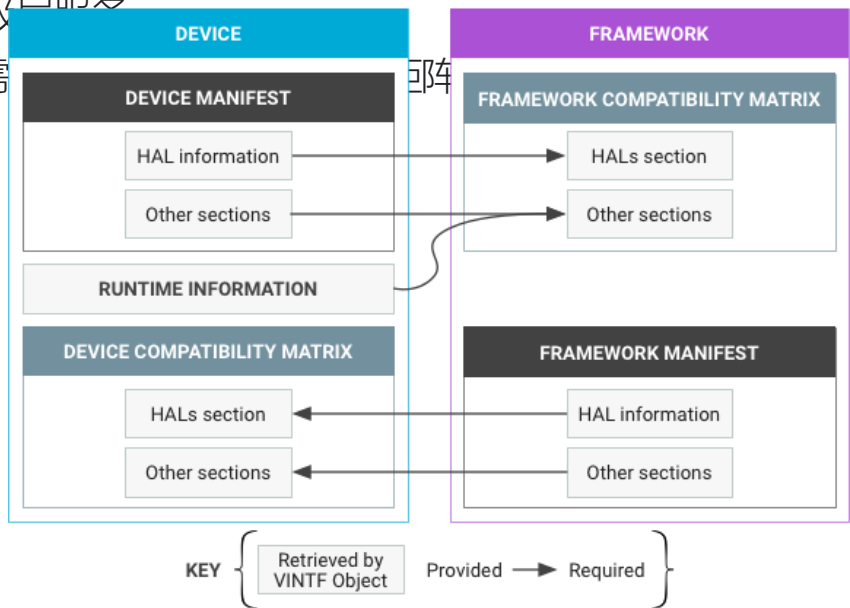
output path

package and package root

## 02 接口定义-设备清单和FCM

VINTF 对象会汇总设备清单和框架清单文件 (XML) 中的数据。这两个清单采用同一种格式，不过并非所有元素都适用于两者。

- **设备清单(Device Manifests)**描述了设备可以为框架提供的静态组件。
- **框架兼容性矩阵FCM**描述了 Android 框架预期从给定设备中获取的内容。此矩阵是一个静态实体，我们会在开发下一个版本的 Android 框架期间手动确定此矩阵的组成。
- **框架清单**描述了框架可以为设备提供的高级服务。
- **设备兼容性矩阵(DCM)**描述了供应商映像期间由开发者手动确定。



## 02 接口定义-设备清单和FCM

对于HIDL接口来说，我们自己定义的设备清单一般会和服务实现放在一起，如下所示：

```
zhilx1@byws013:~/Downloads/android_source/asphalt/vendor/LINUX/android/lenovo/lgsi/interface_impl$ tree battery_impl/
battery_impl/
├── Android.bp
├── Battery.cpp
├── Battery.h
├── service.cpp
├── vendor.lenovo.hardware.battery@1.0-service.rc
└── vendor.lenovo.hardware.battery@1.0-service.xml
0 directories, 6 files
```

```
1 <manifest version="1.0" type="device">
2   <hal format="hidl">
3     <name>vendor.lenovo.hardware.battery</name>
4     <transport>hwbinder</transport>
5     <version>1.0</version>
6     <interface>
7       <name>IBattery</name>
8       <instance>default</instance>
9     </interface>
10   </hal>
11 </manifest>
```

```
1 // FIXME: your file license if you have one
2
3 cc_binary {
4   name: "vendor.lenovo.hardware.battery-service",
5   relative_install_path: "hw",
6   init_rc: ["vendor.lenovo.hardware.battery-service.rc"],
7   vintf_fragments: ["vendor.lenovo.hardware.battery-service.xml"],
8   vendor: true,
9
10  shared_libs: [
11    "liblog",
12    "libcutils",
13    "libdl",
14    "libbase",
15    "libutils",
16    "libhardware",
17    "libbinder_ndk",
18    "vendor.lenovo.hardware.battery-V1-ndk",
19  ],
20
21  srcs: ["service.cpp",
22        "Battery.cpp",
23  ],
24 }
25
```

## 02 接口定义-FCM和设备清单

**Framework compatibility matrix** : 框架兼容性矩阵, 用于指定需要满足哪些框架要求才是合规的供应商实现。兼容性矩阵带有版本编号, 对于每个框架版本, 都会冻结一个新版本的兼容性矩阵。每个框架版本都包含多个FCM。

在 system 分区上发布 FCM 版本的流程:

1. 确保 compatibility\_matrix.F.xml 具有 level="F" 属性。
2. 确保所有设备都已构建并启动。
3. 更新 VTS 测试。
4. 在 vendor 和 qssi 中均需要在 DEVICE\_PRODUCT\_COMPATIBILITY\_MATRIX\_FILE 中添加对应的 FCM 声明。
5. 此外, product 和 system\_ext FCM 也可能列出每个平台 FCM 版本的相应要求。product 和 system\_ext 分区上 FCM 版本的发布分别由这些映像的所有者完成
6. 如果想要废弃 HAL, 则只需要在对应的 FCM 版本中移除即可。

```
hardware/interfaces/compatibility_matrices
├── Android.bp
├── Android.mk
├── build
├── CleanSpec.mk
├── clear_vars.mk
├── compatibility_matrix.3.xml
├── compatibility_matrix.4.xml
├── compatibility_matrix.5.xml
├── compatibility_matrix.6.xml
├── compatibility_matrix.7.xml
├── compatibility_matrix.current.xml
├── compatibility_matrix.empty.xml
├── compatibility_matrix.mk
├── exclude
└── manifest.empty.xml
```

## 02 接口定义-FCM和设备清单

```
28 <compatibility-matrix version="1.0" type="framework" level="6">
29
30 <hal format="hidl" optional="true">
31 <name>vendor.lenovo.hardware.battery</name>
32 <version>1.0</version>
33 <interface>
34 <name>IBattery</name>
35 <instance>default</instance>
36 </interface>
37 </hal>
38
39 </compatibility-matrix>
```

```
1 <manifest version="1.0" type="device">
2 <hal format="hidl">
3 <name>vendor.lenovo.hardware.battery</name>
4 <transport>hwbinder</transport>
5 <version>1.0</version>
6 <interface>
7 <name>IBattery</name>
8 <instance>default</instance>
9 </interface>
10 </hal>
11 </manifest>
```

```
Fetch '/tmp/merge_target_files.qa20pysk/output/PRODUCT/etc/vin
The following HALs in device manifest are not declared in FCM
vendor.lenovo.hardware.battery.IBattery/default (@1)
INCOMPATIBLE
```

```
sdterr:ERROR: files are incompatible: The following instances
vendor.lenovo.hardware.battery.IBattery/default (@1)
Suggested fix:
1. Update deprecated HALs to the latest version.
2. Check for any typos in device manifest or framework compat
3. For new platform HALs, add them to any framework compatib
4. For device-specific HALs, add to DEVICE_FRAMEWORK_COMPATIB
```

Traceback (most recent call last):

```
File "/tmp/tmp.x2QwL0u2X1/bin/merge_target_files/internal/st
File "/tmp/tmp.x2QwL0u2X1/bin/merge_target_files/internal/st
File "/tmp/tmp.x2QwL0u2X1/bin/merge_target_files/_main_.py
File "/tmp/tmp.x2QwL0u2X1/bin/merge_target_files/internal/st
File "/tmp/tmp.x2QwL0u2X1/bin/merge_target_files/internal/st
File "/tmp/tmp.x2QwL0u2X1/bin/merge_target_files/merge_targe
File "/tmp/tmp.x2QwL0u2X1/bin/merge_target_files/merge_targe
File "/tmp/tmp.x2QwL0u2X1/bin/merge_target_files/merge_targe
```

common.ExternalError: Found incompatibilities in the merged ta

```
zhilx1@byws013:~/Downloads/android_source/asphalt/vendor/LINUX/android/lenovo/lgsi$ cat common/components/components.mk | tail -5
#####
# moto input
include $(LGS_I_ROOT)/vendor/interfaces_impl/motinput/motinput.mk
include $(LGS_I_ROOT)/vendor/interfaces_impl/hidlat/hidlat.mk
```

```
zhilx1@byws013:~/Downloads/android_source/asphalt/vendor/LINUX/android/lenovo/lgsi$ cat vendor/interfaces_impl/hidlat/hidlat.mk
$(warning "hidlat.mk")
```

```
DEVICE_PRODUCT_COMPATIBILITY_MATRIX_FILE += \
$(LGS_I_ROOT)/vendor/interfaces_impl/hidlat/1.0/default/compatibility_matrix.xml
```

```
zhilx1@byws013:~/Downloads/android_source/asphalt/vendor/LINUX/android/lenovo/lgsi$ cat vendor/interfaces_impl/hidlat/1.0/default/compatibility_matrix.xml
```

```
<compatibility-matrix version="1.0" type="framework">

<hal format="hidl" optional="true">
  <name>vendor.lenovo.hardware.battery.zhi</name>
  <version>1.0</version>
  <interface>
    <name>IBattery</name>
    <instance>default</instance>
  </interface>
</hal>

</compatibility-matrix>
```

## 问题1

Q1. HIDL接口的哪一项不可以使用hidl-gen生成

- A. c++
- B. service
- C. androidbp





接口概述

接口定义

服务实现

注册和发现服务

版本控制

## 03 服务实现

1. 通过hidl-gen生成Battery.h和Battery.cpp，然后进行完善，并添加对应的rc和xml文件（设备清单）以及至关重要的Android.bp来创建编译规则，否则无法生成对应的service，然后尝试启动该service。

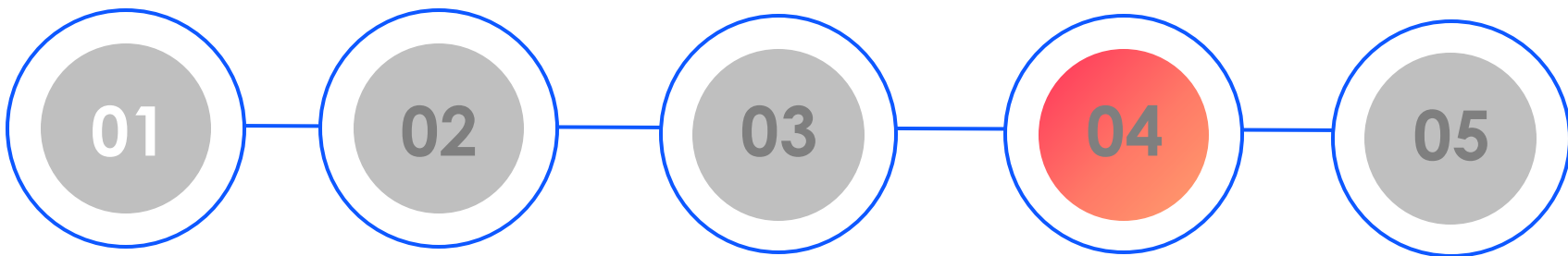
2. 添加sepolicy权限

3. 构建编译规则，将上述服务添加到编译中

```
zhilx1@byws013:~/Downloads/android_source/asphalt/vendor
battery_impl/
├── Android.bp
├── Battery.cpp
├── Battery.h
├── service.cpp
├── vendor.lenovo.hardware.battery@1.0-service.rc
└── vendor.lenovo.hardware.battery@1.0-service.xml
0 directories, 6 files
```

```
28 cc_binary {
29   name: "vendor.lenovo.hardware.battery@1
30   relative_install_path: "hw",
31   proprietary: true,
32   init_rc: ["vendor.lenovo.hardware.batter
33   vintf_fragments: ["vendor.lenovo.hardwa
34   srcs: [
35     "Battery.cpp",
36     "service.cpp",
37   ],
38
39   shared_libs: [
40     "liblog",
41     "libcutils",
42     "libidl",
43     "libbase",
44     "libutils",
45     "libhardware",
46     "libhidlbase",
47     "libhidltransport",
48     "vendor.lenovo.hardware.battery@1.0"
49   ],
50 }
51
```

```
File Edit View Search Terminal Tabs Help
zhilx1@byws013: ~/Downloads/android_source/asphalt/vendor/LINUX/android/lenovo/igs/opensource/int
1 // FIXME: your file license if you have one
2
3 #pragma once
4
5 #include <vendor/lenovo/hardware/battery/1.0/IBattery.h>
6 #include <hidl/MQDescriptor.h>
7 #include <hidl/Status.h>
8
9 namespace vendor::lenovo::hardware::battery::implementation {
10
11 using ::vendor::lenovo::hardware::battery::V1_0::IBattery;
12 using ::android::hardware::hidl_array;
13 using ::android::hardware::hidl_memory;
14 using ::android::hardware::hidl_string;
15 using ::android::hardware::hidl_vec;
16 using ::android::hardware::Return;
17 using ::android::hardware::Void;
18 using ::android::sp;
19
20 struct Battery : public V1_0::IBattery {
21 public:
22   Battery();
23   ~Battery();
24
25   static IBattery* getInstance();
26
27   // Methods from ::vendor::lenovo::hardware::battery::V1_0::IBattery follow.
28   Return<int32_t> getChargeType() override;
29   Return<bool> setUsbSupplyDisabled(bool enable) override;
30   Return<bool> setBatteryChargeDisabled(bool enable) override;
31   Return<bool> setBatteryMaintenanceEnabled(bool enable) override;
32   Return<bool> isBatteryMaintenanceEnabled() override;
33
34   // Methods from ::android::hidl::base::V1_0::IBase follow.
35 private:
36   static Battery* sInstance;
37 };
38
39 // FIXME: most likely delete, this is only for passthrough implementations
40 // extern "C" IBattery* HIDL_FETCH_IBattery(const char* name);
41
42 } // namespace vendor::lenovo::hardware::battery::implementation
```



接口概述

接口定义

服务实现

注册和发现服务

版本控制

## 04 注册和发现服务

### 注册服务

HIDL 接口服务器（实现接口的对象）可注册为已命名的服务。注册的名称不需要与接口或软件包名称相关。如果没有指定名称，则使用名称“默认”。这应该用于不需要注册的服务。接口的两个实现的 HAL。例如，在每个接口中定义的服务

```
status_t status = myFoo->registerAsService();  
status_t anotherStatus = anotherFoo->registerAsService("anotherFoo");
```

HIDL 接口的版本包含在接口本身中。版本自动与服务注册上的方法调用 (android::hardware::IInterface::getInterfaceVersion) 需要注册，并可通过 HIDL 方法参数传递到其他进程，相同 HIDL 方法调用。

```
1 #define LOG_TAG "vendor.lenovo.hardware.battery@1.0-service"  
2  
3 #include <vendor/lenovo/hardware/battery/1.0/IBattery.h>  
4 #include <hidl/HidlSupport.h>  
5 #include <hidl/HidlTransportSupport.h>  
6 #include <android/log.h>  
7 #include "Battery.h"  
8  
9 using vendor::lenovo::hardware::battery::V1_0::IBattery;  
10 using vendor::lenovo::hardware::battery::implementation::Battery;  
11 using android::hardware::configureRpcThreadpool;  
12 using android::hardware::joinRpcThreadpool;  
13 using android::sp;  
14  
15 int main() {  
16  
17     android::sp<IBattery> service = Battery::getInstance();  
18  
19     configureRpcThreadpool(1, true);  
20  
21     if (service != nullptr) {  
22         if (::android::OK != service->registerAsService()) {  
23             return 1;  
24         }  
25     } else {  
26         ALOGE("Can't create instance of Battery, nullptr");  
27     }  
28  
29     joinRpcThreadpool();  
30  
31     return 0;  
32 }  
33  
34
```


C++ 注册HIDL服务

## 04 注册和发现服务

### 发现服务

1. 客户端代码按名称和版本请求给定的接口，并对所需的 HAL 类调用 `getService`。
2. 每个版本的 HIDL 接口都会被视为单独的接口，调用时需要指定名称。
3. 供应商接口对象还会影响所返回接口的传输方法。对于软件包 `vendor.lenovo.hardware.battery@1.0` 中的接口 `IBattery`，`IBattery::getService` 返回的接口始终使用设备清单中针对 `vendor.lenovo.hardware.battery` 声明的传输方法；如果该传输方法不存在，则返回 `nullptr`。
4. 在某些情况下，即使没有获得相关服务，也可能需要立即继续。

```
private static void getIBatteryService() {  
    try {  
        IBattery_v2 = vendor.lenovo.hardware.battery.V2_0.IBattery.getService(); // get IBattery@2.0 service  
        IBattery_v2.linkToDeath(mDeathRecipient, 0);  
    } catch (Exception e) {  
        Slog.e(TAG, "Failed to get IBattery Service V2.0: " + e);  
        try {  
            IBattery_v1 = vendor.lenovo.hardware.battery.V1_0.IBattery.getService(); // get IBattery@1.0 service  
            IBattery_v1.linkToDeath(mDeathRecipient, 0);  
        } catch (Exception e2) {  
            Slog.e(TAG, "Failed to get IBattery Service V1.0: " + e2);  
        }  
    }  
}
```



## 04 注册和发现服务

### 终止服务

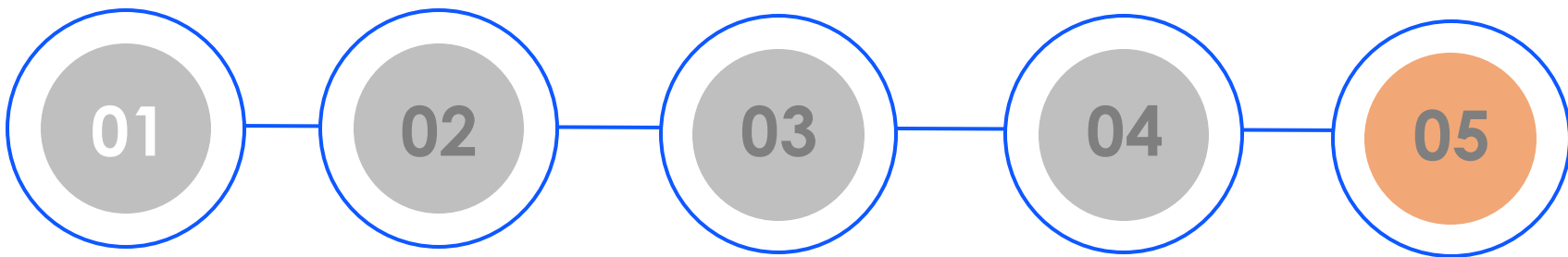
想要在服务终止时收到通知的客户端会接收到框架传送的终止通知。如需接收通知，客户端必须：

1. 创建 HIDL 类/接口 `hidl_death_recipient` 的子类（在 C++ 代码中，而不是在 HIDL 中）。
2. 替换其 `serviceDied()` 方法。
3. 实例化 `hidl_death_recipient` 子类的对象。
4. 在要监控的服务上调用 `linkToDeath()` 方法，并传入 `IDeathRecipient` 的接口对象。

伪代码示例（C++ 和 Java 类似）：

```
class IMyDeathReceiver : hidl_death_recipient {  
    virtual void serviceDied(uint64_t cookie,  
        wp<IBase>& service) override {  
        log("RIP service %d!", cookie); // Cookie should be 42  
    }  
};  
....  
IMyDeathReceiver deathReceiver = new IMyDeathReceiver();  
m_importantService->linkToDeath(deathReceiver, 42);
```

```
try {  
    mBatteryService = IBattery.getService(true); // get service  
    mBatteryService.linkToDeath(mDeathRecipient, 0);  
    Log.d(TAG, msg: "mBatteryService:" + mBatteryService);  
} catch (Exception e) {  
    Log.d(TAG, msg: "Failed to get IBattery Service" + e);  
}  
  
private IHwBinder.DeathRecipient mDeathRecipient = new IHwBinder.DeathRecipient() {  
    @Override  
    public void serviceDied(long l) {  
        Log.e(TAG, msg: "IBattery serviceDied");  
        try {  
            mBatteryService.unlinkToDeath(mDeathRecipient);  
        } catch (RemoteException e) {  
            Log.e(TAG, msg: "IBattery service unlink death failed", e);  
        }  
        mBatteryService = null;  
    }  
};
```



接口概述

接口定义

服务实现

注册和发现服务

版本控制

## 05 版本控制

HIDL 要求每个使用 HIDL 编写的接口均必须带有版本编号。HAL 接口一经发布便会被冻结，如果要做任何进一步的更改，都只能在接口的新版本中进行。虽然无法对指定的已发布接口进行修改，但可通过其他接口对其进行扩展。

针对指定版本（如 `vendor.Lenovo.hardware.battery`）发布后，HIDL 软件包（如 `1.0`）便不可再改变；我们无法对其进行更改。如果要对已发布软件包中的接口进行修改，或要对其 UDT 进行任何更改，都只能在另一个软件包中进行。

从概念上来讲，软件包可通过以下方式之一与另一个软件包相关：

- 完全不相关。
- 软件包级向后兼容的可扩展性：这是软件包的一种 `mirror` 版本升级。
- 扩展原始的向后不兼容性。这是软件包的一种 `major` 版本升级，并且新旧两个版本之间不需要存在任何关联。如果存在关联，这种关联可以通过以下方式来表示：组合旧版本软件包中的类型，以及继承旧软件包中的部分接口。

同时，当软件包满足版本继承和控制要求时，`hidl-gen`会强制执行向后兼容性规则。



## 06 小结

1. 创建xxx.hal，如IBattery.hal，编写完成后，可以使用mmm命令编译一下，看看能不能生成对应的java客户端  
首先可以先编译一下看看能不能生成对应的服务：mmm <目录名>。

```
mmm lenovo/lgsi/opensource/interfaces/battery
```

2. 使用hidl-gen生成Android.bp。
3. 使用hidl-gen生成current值。
4. 使用hidl-gen生成Battery.h和Battery.cpp，然后进行完善，并添加对应的rc和xml文件以及至关重要的Androd.bp来创建编译规则，然后尝试启动该service。
5. 添加sepolicy权限，首先可以先使用lunch命令，然后查看  
/lenovo/lgsi/vendor/components/gen/feature\_sepolicy\_vendor.mk中是否将sepolicy的目录包含进去，

然后尝试通过make selinux\_sepolicy来看看是否报错  
编译完成后，再编译vendor或者整编验证该服务是否可以正常自启以及功能是否可用。

## 06 小结

6. 需要注意的是由于HAL接口需要在vendor和qssi中通信，因此每次发布之前需要先将接口冻结freeze，然后并生成对应的FCM(Framework Compatibility Matrix)，FCM分别需要在qssi和vendor目录下定义，然后先编译qssi再编译vendor，否则会报“The following instances are in the device manifest bug not specified in framework compatibility matrix”。

在lgsl中可以采用下面的定义：

- 在service的实现中定义对应的FCM
- 在当前目录中定义一个mk文件，将对应的FCM声明到DEVICE\_PRODUCT\_COMPATIBILITY\_MATRIX\_FILE中
- 在lenovo/lgsi/common/comontents/components.mk中将上述mk文件声明进去，从而实现在编译的时候可以调到
- 需要注意的是，如果想要服务正常启动，则需要在vendor分区中添加FCM，并且需要将compatibility-matrix 的type标记为framework类型，否则会导致编译失败。

7. 在高通平台，hidl服务的实现还需要在qiifa中进行定义，否则会报QIIFA的错误。

vendor/qcom/proprietary/commonsys-intf/QIIFA-cmd/中

8. 刷机验证，查看对应服务是否成功启动，如果仍然存在sepolicy问题，则根据报错——添加对应规则即可。

## 问题2

Q2. HIDL接口发布后如何进行扩展升级(多选)

- A. 在原接口上直接修改
- B. 创建一个新的无关的接口
- C. 实现一个新的软件包版本



03

# AIDL接口定义 和实现

# AIDL实现流程



## 接口概述

AIDL接口的描述、与HIDL接口的差异

## 接口定义

AIDL接口的添加、编译规则设置、设备清单和FCM兼容性矩阵

## 服务实现

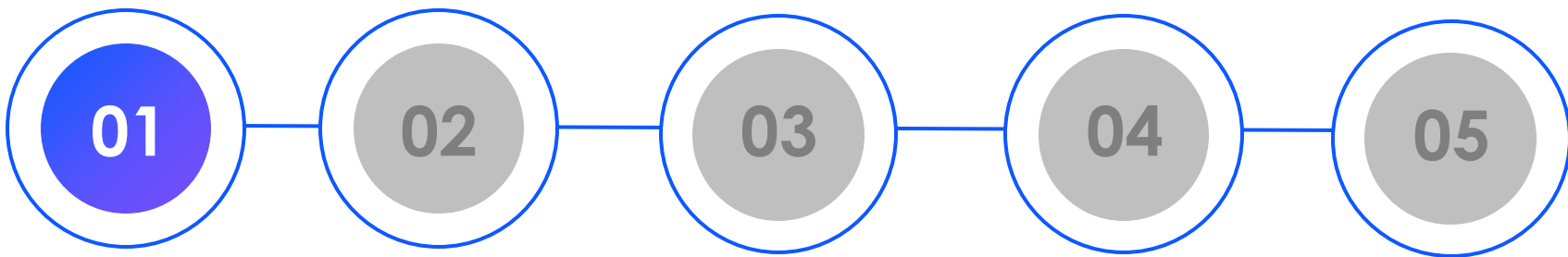
服务实现的步骤详解和SEPOLICY描述

## 注册和发现服务

服务的注册、发现和调用

## 版本控制

接口扩展和更新



接口概述

接口定义

服务实现

注册和发现服务

版本控制

## 01 AIDL接口描述

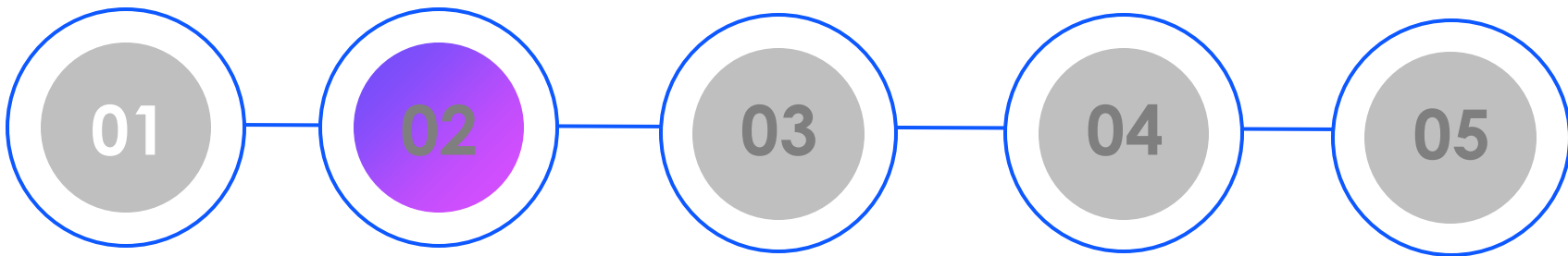
Android 接口定义语言 (AIDL) 是一种让用户抽象出 IPC 的工具。给定一个接口，各种构建系统使用aidl来构建 C++ 或 Java 绑定，以便该接口可以跨进程使用。

Android 11 中引入了在 Android 中使用 AIDL 实现 HAL 的功能。这样就能在不使用 HIDL 的情况下实现 Android 的部分代码。在可能的情况下，应将 HAL 转换为仅使用 AIDL。

如果 HAL 使用 AIDL 在框架组件（例如 system.img 中的组件）和硬件组件（例如 vendor.img 中的组件）之间进行通信，必须使用稳定的 AIDL。不过，如需在分区内进行通信（例如从一个 HAL 到另一个 HAL），则对需要使用的 IPC 机制没有任何限制。

AIDL优点：

- 调试、优化和保护简单
- 有内建的版本控制机制
- 后续持续开发方便
- 扩展方便



接口概述

接口定义

服务实现

注册和发现服务

版本控制



## 02 接口定义-适用于HAL的AIDL

在实现AIDL接口定义分为以下几个步骤：

### 1. 定义新的AIDL接口

1.1 如果HIDL接口存在，则可以使用hidl2aidl工具直接转换（如果不存在，则需要自己定义Andorid.bp文件）

1.2 执行转换

1.3 调整编译规则(Andorid.bp)

### 2. 向Vendor中添加AIDL接口

2.1 使用update-api freeze-api管理接口版本

2.2 配置Framework Compatibility Matrix(FCM, 兼容性矩阵)

2.3 AIDL编译配置

### 3. 实现service

3.1 编写service

3.2 创建service编译规则

3.3 将service添加进系统

### 4. 配置开机自启和Sepolicy权限

4.1 配置SEPOLICY权限

4.2 将该服务添加进feature中

### 5. java层调用service



接口概述

接口定义

服务实现

注册和发现服务

版本控制

## 03 服务实现-添加AIDL接口

以vendor.lenovo.hardware.battery为例进行介绍

1. 创建aidl文件或者使用hidl2aidl生成对应的aidl文件和Android.bp文件
2. 使用update-api和 freeze-zpi来管理接口版本，这个工具做的事情就是：把我们写的接口复制一份，然后标上版本号，作为一个冻结的(stable)版本。之后正式编译时，就自动使用它复制出来的版本。

```
source <项目位置>/build/envsetup.sh
lunch 项目名-userdebug
m vendor.lenovo.hardware.battery-update-api // 复制现在的版本到aidl_api/current
m vendor.lenovo.hardware.battery-freeze-api // 从current生成一个新的版本(号)
```

```
zhilx1@byws013:~/Downloads/android_source/lsu-u/lenovo/lsu/opensource/interfaces/battery/aidl$ tree .
```

```
.
├── aidl_api
│   ├── vendor.lenovo.hardware.battery
│   │   ├── 1
│   │   │   ├── vendor
│   │   │   │   ├── lenovo
│   │   │   │   │   ├── hardware
│   │   │   │   │   │   ├── battery
│   │   │   │   │   │   │   └── IBattery.aidl
│   │   └── current
│   │       ├── vendor
│   │       │   ├── lenovo
│   │       │   │   ├── hardware
│   │       │   │   │   ├── battery
│   │       │   │   │   │   └── IBattery.aidl
│   └── Android.bp
├── vendor
│   ├── lenovo
│   │   ├── hardware
│   │   │   ├── battery
│   │   │   │   └── IBattery.aidl
```

```
./battery/
├── 1.0
│   ├── Android.bp
│   └── IBattery.hal
└── 2.0
    ├── Android.bp
    └── IBattery.hal
```

## 03 服务实现-添加AIDL接口

HIDL

```
./battery/
├── 1.0
│   ├── Android.bp
│   └── IBattery.hal
├── 2.0
│   ├── Android.bp
│   └── IBattery.hal
```

```
1 package vendor.lenovo.hardware.battery@1.0;
2
3 interface IBattery{
4     /**
5      * 0表示正常, 1表示PD_PPS, 2表示FLOAT类型
6      */
7     getChargerType() generates(int32_t type);
8
9     /**
10    * 该方法用来判断是否disable usb供电
11    * 当setUsbSupplyDisabled(1)时, pmic usbin 高阻模式, 不为系统供电
12    * 当setUsbSupplyDisabled(0)时, usb恢复系统供电
13    */
14    setUsbSupplyDisabled(bool enable) generates(bool success);
15
16    /**
17    * 该方法用来判断是否disable 充电
18    * 当setBatteryChargeDisabled(1)时, 充电disable
19    * 当setBatteryChargeDisabled(0)时, 充电enable, 根据充电功能, 该启动快充的要启动快充 (如充电本身的功能E
20    */
21    setBatteryChargeDisabled(bool enable) generates(bool success);
22
23    setBatteryMaintenanceEnabled(bool enable) generates(bool success);
24    isBatteryMaintenanceEnabled() generates(bool success);
25 };
```

AIDL

```
zhilixi@byws013:~/Downloads/android_source/lsqi-u/lenovo/lsqi/opensource/interfaces/battery/aidl$ tree .
.
├── aidl_api
│   └── vendor.lenovo.hardware.battery
│       └── 1
│           └── vendor
│               └── lenovo
│                   └── hardware
```

File Edit View Search terminal Help

1 // This is the expected build file, but it may not be right in all cases

```
3 aidl_interface {
4     name: "vendor.lenovo.hardware.battery",
5     vendor_available: true, 这里需要设置为true
6     owner: "Lenovo", 需要添加owner, 否则可能会编译失败
7     // host_supported: true,
8     //frozen: true,
9     srcs: ["vendor/lenovo/hardware/battery/*.aidl"],
10    stability: "vintf", 需要添加vintf, 指定为稳定的, 一般会自动生成
11    backend: {
12        cpp: {
13            // FIXME should this be disabled?
14            // prefer NDK backend which can be used anywhere
15            // If you disable this, you also need to delete the C++
16            // translate code.
17            enabled: false,
18        },
19        java: {
20            sdk_version: "system_current",
21        },
22        ndk: {
23            enabled: true, 如果使用ndk, 则需要指定为true
24        },
25    },
26    versions: [
27        "1",
28        "2",
29    ],
30
31 }
```

无需自己添加, 会通过update-api自动生成,  
并且Android U的版本标记有所不同

## 03 服务实现-配置FCM兼容性矩阵

FCM 指定了qssi与vendor间的兼容关系。因此，对新定义的接口，需要在qssi和vendor两个目录中都配置FCM兼容关系。

在vendor和qssi目录中添加下面的FCM

```
<hal format="hidlaidl" optional="true">
  <name>vendor.lenovo.hardware.battery</name>
  <version>1.0</version>
  <interface>
    <name>IBattery</name>
    <instance>default</instance>
  </interface>
</hal>
```


在system.img中我们希望既支持AIDL，同时又对之前的HIDL接口支持。因此在QSSI中只需要添加AIDL接口的配置到FCM中即可。

## 03 服务实现-配置Device Manifest文件

上面对vendor.img中的FCM进行了修改，去除了HIDL接口，只保留AIDL。因此需要先找到之前HIDL接口编译的manifest，然后将其修改为aidl。

例如vendor.lenovo.hardware.battery在 vendor/LINUX/android/lenovo/lgsi/interface\_impl/battery\_impl下将 vendor.lenovo.hardware.battery.xml文件中， 其内容如下：

```
<manifest version="1.0" type="device">
  <hal format="hidl">
    <name>vendor.lenovo.hardware.battery</name>
    <transport>hwbinder</transport>
    <version>1</version>
    <interface>
      <name>IBattery</name>
      <instance>default</instance>
    </interface>
  </hal>
</manifest>
```



```
<manifest version="1.0" type="device">
  <hal format="aidl">
    <name>vendor.lenovo.hardware.battery</name>
    <version>1</version>
    <!-- 这里也可以使用上面的写法，不过最新的aidl改为了如下的写法 -->
    <fqname>IBattery/default</fqname>
  </hal>
</manifest>
```

到这里为止，便可以尝试编译aidl服务，可以尝试先编译qssi再编译vendor，然后看看是否生成对应的aidl服务。

## 03 服务实现-Service实现

首先在aidl目录下，创建一个default目录。我们所有的service相关内容都会放到这个default目录下。

理论上，service可以放在任何地方，但是和aidl放在一起比较好，这边方便管理，如IBattery.aidl如下所示：

```
zhilx1@byws013:~/Downloads/android_source/asphalt/vendor/LINUX/android/lenovo/lgsi/opensource/interfaces/battery/aidl$ tree .
.
├── aidl_api
│   └── vendor.lenovo.hardware.battery
│       ├── 1
│       │   ├── vendor
│       │   │   ├── lenovo
│       │   │   │   ├── hardware
│       │   │   │   └── battery
│       │   │       └── IBattery.aidl
│       └── current
│           └── vendor
│               ├── lenovo
│               │   ├── hardware
│               │   └── battery
│               └── IBattery.aidl
├── Android.bp
├── battaidl.mk
├── default
│   ├── Android.bp
│   ├── Battery.cpp
│   ├── Battery.h
│   ├── compatibility_matrix.3.xml
│   ├── compatibility_matrix.5.xml
│   ├── compatibility_matrix.6.xml
│   ├── compatibility_matrix.xml
│   ├── main.cpp
│   ├── vendor.lenovo.hardware.battery-service.rc
│   └── vendor.lenovo.hardware.battery-service.xml
└── vendor
    ├── lenovo
    │   ├── hardware
    │   └── battery
    │       └── IBattery.aidl
```

## 03 服务实现-Service实现

实现service，只需要按照上述截图中的内容实现即可。Battery.h主要为了扩展aidl接口提供的Bn类，扩展为

具体来说，我们需要对aidl中的每一个接口，都进行

因为上一节中已经成功编译了aidl，红框中的部分可以直接从其中复制过来，对应的文件路径为：

out/soong/.intermediates/xxx<项目路径>  
下，如：

out/soong/.intermediates/lenovo  
/lgsi/opensource/interfaces/bat  
tery/aidl/vendor.lenovo.hardwar  
e.battery-V2-ndk\_platform-  
source/gen/include/aidl/vendor/  
lenovo/hardware/battery/BpBattt  
ery.h

```
// Lenovo/Lgsi/opensource/interfaces / battery/aidl/default/Battery.h
/**
 * IBattery.aidl impl
 */
#pragma once

#include <android-base/file.h>
#include <utils/Log.h>
#include <aidl/vendor/lenovo/hardware/battery/BnBattery.h>

namespace aidl {
namespace vendor {
namespace lenovo {
namespace hardware {
namespace battery {

using ::android::base::ReadFileToString;
using ::android::base::WriteStringToFile;
using ::std::string;
using ::std::shared_ptr;
using ::ndk::ScopedAStatus;

class Battery : public BnBattery {
public:
    Battery() = default;
    virtual ~Battery();

    // 这里为aidl中对应的方法，需要使用ndk进行实现
    ::ndk::ScopedAStatus getBatteryMaintenanceEnabledV2(bool* _aidl_return) override;
    ::ndk::ScopedAStatus getChargerType(int32_t* _aidl_return) override;
    ::ndk::ScopedAStatus isBatteryMaintenanceEnabled(bool* _aidl_return) override;
    ::ndk::ScopedAStatus setBatteryChargeDisabled(bool in_enable, bool* _aidl_return) override;
    ::ndk::ScopedAStatus setBatteryMaintenanceEnabled(bool in_enable, bool* _aidl_return) override;
    ::ndk::ScopedAStatus setBatteryMaintenanceEnabledV2(bool in_enable, bool* _aidl_return) override;
    ::ndk::ScopedAStatus setUsbSupplyDisabled(bool in_enable, bool* _aidl_return) override;

};

} // battery
} // hardware
} // lenovo
} // vendor
} // aidl
```



## 03 服务实现-创建service编译规则

要配置编译规则，需要添加对应的Android.bp, .rc和.xml文件。

1. Andorid.bp：该文件用于指定service编译规则。
2. rc文件，如vendor.lenovo.hardware.battery-service.rc，该文件可以配置为开机自启
3. xml文件，如vendor.lenovo.hardare.battery.xml，该文件与前面介绍的manifest相关。
4. 将该service添加进系统。然后编译即可。

```
cc_binary {
    name: "vendor.lenovo.hardware.battery-service",
    relative_install_path: "hw",
    init_rc: ["vendor.lenovo.hardware.battery-service.rc"],
    vintf_fragments: ["vendor.lenovo.hardware.battery-service.xml"],
    // vendor: true,
    shared_libs: [
        "libbase",
        "libbinder_ndk", // Stable aidl必须
        "libcutils",
        "liblog",
        "libutils",
        "vendor.lenovo.hardware.battery-V2-ndk", // 包名, 在Andoird 12以下版本, 需要使用ndk_platform来替换ndk字符。
    ],
    srcs: [
        "main.cpp",
        "Battery.cpp",
    ],
}

filegroup {
    name: "vendor.lenovo.hardware.battery-service.xml",
    srcs: ["vendor.lenovo.hardware.battery-service.xml"],
}

filegroup {
    name: "vendor.lenovo.hardware.battery-service.rc",
    srcs: ["vendor.lenovo.hardware.battery-service.rc"],
}
```

// service + 注册名 (可以随便起) + 编译后的模块所在路径

```
service lenovo_aidl_battery_default /vendor/bin/hw/vendor.lenovo.hardware.battery-service
class hal
user root # 这里需要指定为root用户, 否则可能会出现权限问题
group system # 最好指定为root用户, 否则可能会出现权限问题
```

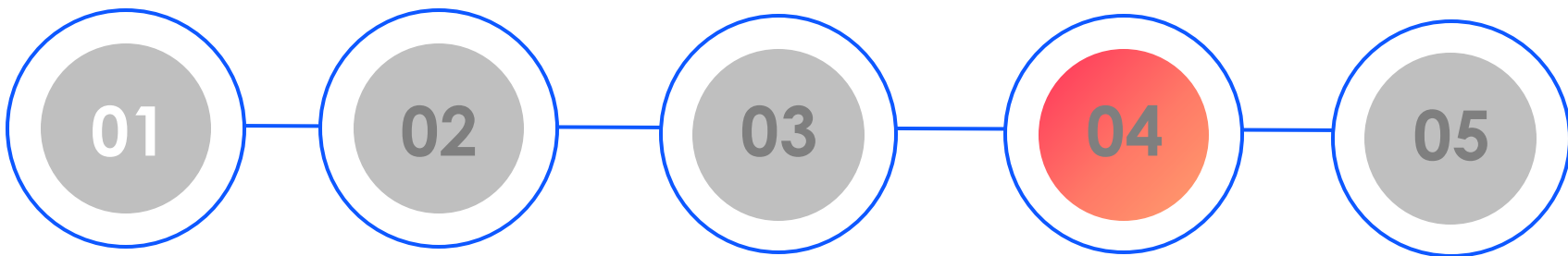
## 03 服务实现-Sepolicy配置

为了确保上述service可以开机自启，则还需要配置相关的SEPOLICY权限。

SEpolicy是引入了SELinux (kernel) 后所需求的一种安全策略。SELinux会默认阻止一些关键操作（例如文件操作，ioctl等），我们需要逐条声明后，才能合法进行这些操作。

1. 创建sepolicy目录
2. 在该目录中添加：attribute、file\_contexts、hwservice.te、hwservice\_contexts等

```
zhilx1@byws013:~/Downloads/android_source/asphalt/vendor/LINUX/android/lenovo/lgsi/vendor/components/sepolicy/batteryaidl$ tree .
.
├── private
│   ├── attributes
│   ├── file_contexts
│   ├── hwservice_contexts
│   ├── hwservice.te
│   ├── lenovo_aidl_battery_default.te
│   ├── lenovo_aidl_battery.te
│   ├── service_contexts
│   ├── service.te
│   └── system_server.te
```



接口概述

接口定义

服务实现

注册和发现服务

版本控制

## 04 注册和发现服务

AIDL注册和发现服务的实现方法与HIDL大同小异，区别在于：

1. AIDL服务的所有操作都需要先调用asBinder()进行转换
2. 注册时需要指定AIDL服务descriptor。

这里可以看一下IBattery的注册过程：

```
zhilix1@byws013: ~/Downloads/android_source/asphalt/vendor/LINUX/android/lenovo/lgsi/opensource/interfaces/ba... x zhilix1@byws0
1 // Copyright (C) 2023 Lenovo
2 //
3 //
4 #include <android-base/logging.h>
5 #include <android/binder_manager.h>
6 #include <android/binder_process.h>
7
8 #include "Battery.h"
9
10 using ::aidl::vendor::lenovo::hardware::battery::Battery;
11
12 int main() {
13     LOG(INFO) << "Battery AIDL starting up ---";
14
15     ABinderProcess_setThreadPoolMaxThreadCount(1);
16     std::shared_ptr<Battery> battery = ndk::SharedRefBase::make<Battery>();
17
18     const std::string instance = std::string() + Battery::descriptor + "/default";
19     binder_status_t status = AServiceManager_addService(battery->asBinder().get(), instance.c_str());
20     CHECK(status == STATUS_OK);
21
22     ABinderProcess_joinThreadPool();
23
24     LOG(INFO) << "Battery AIDL Join Thread pool ---";
25     return 0;
26 }
```

1. 创建线程池和指针

2. 启动该服务，然后将其加入到线程池中

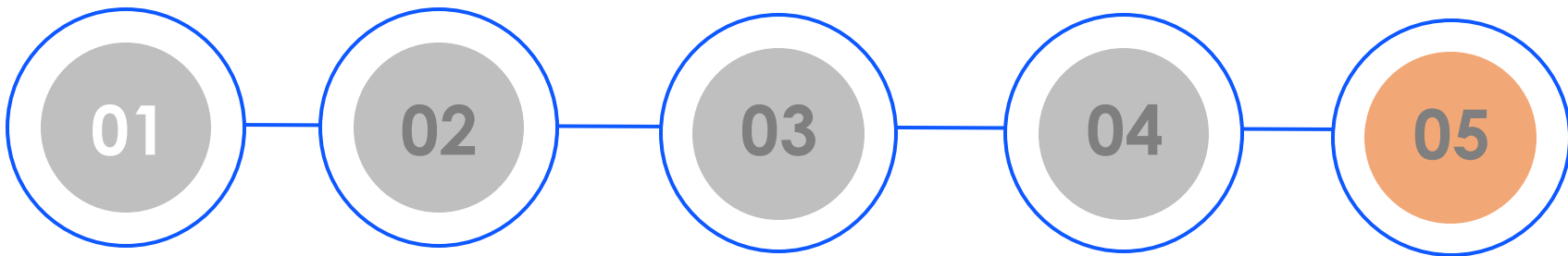
## 04 注册和发现服务

系统层级对AIDL服务发现和调用:

1. 在QSSI/MSSI中创建接口定义
2. 生成对应的jar包
3. 引用jar包, 然后调用即可

```
zhilixi@byws013:~/Downloads/android_source/lsqi-u/lenovo/lsqi/opensource/interfaces/battery$ tree .
.
├── 1.0
│   ├── Android.bp
│   └── IBattery.hal
├── 2.0
│   ├── Android.bp
│   └── IBattery.hal
└── aidl
    ├── aidl_api
    │   └── vendor.lenovo.hardware.battery
    │       ├── 1
    │       │   ├── vendor
    │       │   │   ├── lenovo
    │       │   │   │   ├── hardware
    │       │   │   │   │   ├── battery
    │       │   │   │   │   │   └── IBattery.aidl
    │       │   └── current
    │       │       ├── vendor
    │       │       │   ├── lenovo
    │       │       │   │   ├── hardware
    │       │       │   │   │   ├── battery
    │       │       │   │   │   │   └── IBattery.aidl
    │       └── Android.bp
    │           ├── vendor
    │           │   ├── lenovo
    │           │   │   ├── hardware
    │           │   │   │   ├── battery
    │           │   │   │   │   └── IBattery.aidl
```

```
// hal interface static libs
java_library {
    name: "lsqi-hal-interface-libs",
    static_libs: [
        "vendor.lenovo.hardware.battery-V2.0-java",
        "vendor.lenovo.hardware.battery-V1-java", // AIDL
        "vendor.lenovo.hardware.display-V1.0-java",
        "vendor.lenovo.hardware.display-V1-java", // AIDL
        "vendor.lenovo.hardware.factory-V1.0-java",
        "vendor.lenovo.hardware.factory-V1-java", // AIDL
        "vendor.lenovo.hardware.keyboard-V1.0-java",
        "vendor.lenovo.hardware.keyboard-V1-java", // AIDL
        "vendor.lenovo.hardware.misc-V1.0-java",
        "vendor.lenovo.hardware.misc-V1-java", // AIDL
        "vendor.lenovo.hardware.performance-V1.0-java",
        "vendor.lenovo.hardware.performance-V1-java", // AIDL
        "vendor.lenovo.hardware.ifaa-V2.0-java",
        "vendor.lenovo.hardware.soter-V1.0-java",
        "vendor.lenovo.hardware.touchscreen-V1.1-java",
        "vendor.lenovo.hardware.touchscreen-V1-java", // AIDL
    ],
    sdk_version: "core_platform",
    installable: false,
}
```



接口概述

接口定义

服务实现

注册和发现服务

版本控制

## 05 版本控制

如vendor.lenovo.hardware.battery-v1的aidl接口无法满足要求时，我们需要对其进行扩展才行。

在HIDL接口中，对接口的扩展会涉及到UDT版本号更新，如vendor.lenovo.hardware.battery@1.0，扩展或升级之后接口会变为vendor.lenovo.hardware.battery@1.1，因此system.img中上层的调用也需要同时修改。由于HIDL中接口名称发生了改变，因此还需要实现相关service，可以发现修改较为复杂。

而在AIDL中，相比于HIDL，扩展升级则较为简单：

1. 首先对.aidl文件中进行扩展
2. 调用m vendor.lenovo.hardware.battery-update-api和m vendor.lenovo.hardware.battery-freeze-api来生成对应的api和freeze版本，此时我们会发现Android.bp中的版本号会从1升级为2，同时会在对应的目录下生成2.0的aidl文件
3. 对于vendor而言，此时需要在service中进行修改，对相关服务进行扩展即可，无需重新创建一份service对象
4. 最后修改FCM和DM文件版本即可
5. 对于system而言，需要将import的包从1.0升级为2.0，然后调用相关方法即可

## 05 版本控制

```
zhilx1@byws013:~/Downloads/android_source/lsi-u/lenovo/lsi/opensource/interfaces/battery/aidl$ git diff
diff --git a/battery/aidl/Android.bp b/battery/aidl/Android.bp
index 7a7076c..3d7664f 100644
--- a/battery/aidl/Android.bp
+++ b/battery/aidl/Android.bp
@@ -28,6 +28,11 @@ aidl interface {
     version: "1",
     imports: [],
 }
+ {
+     version: "2",
+     imports: [],
+ },
+ ],
}

diff --git a/battery/aidl/aidl_api/vendor.lenovo.hardware.battery.current/vendor/lenovo/hardware/battery/IBattery.aidl b/battery/aidl/aidl_api/vendor.le
/battery/IBattery.aidl
index 8449164..0da65e9 100644
--- a/battery/aidl/aidl_api/vendor.lenovo.hardware.battery.current/vendor/lenovo/hardware/battery/IBattery.aidl
+++ b/battery/aidl/aidl_api/vendor.lenovo.hardware.battery.current/vendor/lenovo/hardware/battery/IBattery.aidl
@@ -28,4 +28,5 @@ interface IBattery {
    boolean setBatteryMaintenanceEnabledV2(in boolean enable);
    boolean setUsbSupplyDisabled(in boolean enable);
    boolean setShipModeState(in int shipMode);
    int getType();
+ }

diff --git a/battery/aidl/vendor/lenovo/hardware/battery/IBattery.aidl b/battery/aidl/vendor/lenovo/hardware/battery/IBattery.aidl
index 1687051..b5ba034 100644
--- a/battery/aidl/vendor/lenovo/hardware/battery/IBattery.aidl
+++ b/battery/aidl/vendor/lenovo/hardware/battery/IBattery.aidl
@@ -53,4 +53,6 @@ interface IBattery {
    * return: 设置是否成功
    */
    boolean setShipModeState(in int shipMode);
+ }
+ int getType();
+ }
```

```
zhilx1@byws013:~/Downloads/android_source/lsi-u/lenovo/lsi/opensource/interfaces/battery/aidl$ tree .
.
├── aidl_api
│   ├── vendor.lenovo.hardware.battery
│   │   ├── 1
│   │   │   ├── vendor
│   │   │   │   ├── lenovo
│   │   │   │   │   ├── hardware
│   │   │   │   │   │   ├── battery
│   │   │   │   │   │   │   IBattery.aidl
│   │   │   └── 2
│   │   │       ├── vendor
│   │   │       │   ├── lenovo
│   │   │       │   │   ├── hardware
│   │   │       │   │   │   ├── battery
│   │   │       │   │   │   │   IBattery.aidl
│   │   └── current
│   │       ├── vendor
│   │       │   ├── lenovo
│   │       │   │   ├── hardware
│   │       │   │   │   ├── battery
│   │       │   │   │   │   IBattery.aidl
├── Android.bp
└── vendor
    ├── lenovo
    │   ├── hardware
    │   │   ├── battery
    │   │   │   IBattery.aidl
```



## 问题3

Q3. 下面哪一项是AIDL接口的优点(多选)?

- A. 调试、优化简单
- B. 系统层调用简单
- C. 具有较优的版本控制系统



04

# HAL接口 的兼容调用



## 接口调用

HIDL/AIDL在framework/java层调用流程是类似的。

1. 首先需要在调用的Andorid.bp中声明需要调用的服务命令：

```
static_libs: [  
    "vendor.lenovo.hardware.battery-V1.0-java" // hidl服务  
    "vendor.lenovo.hardware.battery-V1-java", // aidl服务 ],
```

2. 在代码中import该包
3. 实例化HIDL/AIDL对象
4. 调用HIDL/AIDL中的方法
5. 如果App想要调用系统级别的HAL接口，可以直接将HIDL/AIDL的jar包导入到apk中，然后直接引用即可。
6. 在注册服务时，建议同时监听HIDL、AIDL service的death情况，如果判断到服务死亡，则客户端应该主动断开然后重新注册，避免上层软件不生效的情况。

# 接口调用

```
211 interface IBatteryService {
212     public int getChargerType();
213     public boolean setUsbSupplyDisabled(boolean flag);
214     public boolean setBatteryChargeDisabled(boolean flag);
215     public boolean setBatteryMaintenanceEnabled(boolean flag);
216     public boolean setBatteryMaintenanceEnabledV2(boolean flag);
217     public boolean isBatteryMaintenanceEnabled();
218 }
219
220 /**
221  * Create a class to call the IBattery Service Manager, thereby shielding
222  * differences in framework code and reducing code modifications.
223  *
224  * @file: IBatteryServiceManager.java
225  * @author: zhixi@lenovo.com
226  * @date: 2023/07/26
227  * @version: 1.1 (add aidl interface)
228  *
229  * @hide
230  */
231 public class IBatteryServiceManager implements IBatteryService {
232     private static final String TAG = "IBatteryServiceManager";
233
234     private static final String BATTERY_SERVICE_NAME = IBattery.DESRIPTOR + "/default";
235
236     private static vendor.lenovo.hardware.battery.V1_0.IBattery IBatteryHAL_v1; // HIDL V1
237     private static vendor.lenovo.hardware.battery.V2_0.IBattery IBatteryHAL_v2; // HIDL V2
238     private static vendor.lenovo.hardware.battery.IBattery mBatteryAidl; // AIDL
239
240     private final Object mLock = new Object();
241
242     private IBatteryServiceManager() {
243         getIBatteryService();
244     }
245
246     private volatile static IBatteryServiceManager sInstance;
247     public static IBatteryServiceManager getInstance() {
248         if (sInstance == null) {
249             synchronized (IBatteryServiceManager.class) {
250                 if (sInstance == null) {
251                     sInstance = new IBatteryServiceManager();
252                 }
253             }
254         }
255         return sInstance;
256     }
257 }
```

建议基类设为单例模式  
避免重复创建

```
194 private static void getIBatteryService() {
195     if (mBatteryAidl != null || IBatteryHAL_v1 != null || IBatteryHAL_v2 != null) return;
196     try {
197         mBatteryAidl = IBattery.Stub.asInterface(ServiceManager.waitForDeclaredService(BATTERY_SERVICE_NAME)); // get AIDL service
198         if (mBatteryAidl == null) IBatteryHAL_v2 = vendor.lenovo.hardware.battery.V2_0.IBattery.getService(); // get IBattery@2.0 service
199     } catch (Exception e) {
200         Slog.e(TAG, "Failed to get IBattery Service V2.0," + e);
201         try {
202             IBatteryHAL_v1 = vendor.lenovo.hardware.battery.V1_0.IBattery.getService(); // get IBattery@1.0 service
203         } catch (Exception e2) {
204             Slog.e(TAG, "Failed to get IBattery Service V1.0," + e2);
205         }
206     }
207     Slog.d(TAG, "mBatteryAidl = " + mBatteryAidl + ", IBatteryHAL_v2 = " + IBatteryHAL_v2 + ", IBatteryHAL_v1 = " + IBatteryHAL_v1);
208 }
209 }
```

```
197 private static void getIBatteryService() {
198     if (mBatteryAidl != null || IBatteryHAL_v1 != null || IBatteryHAL_v2 != null) return;
199     try {
200         mBatteryAidl = IBattery.Stub.asInterface(ServiceManager.waitForDeclaredService(BATTERY_SERVICE_NAME)); // get AIDL service
201         if (mBatteryAidl != null) mBatteryAidl.asBinder().linkToDeath(mAidlDeathRecipient, 0);
202         if (mBatteryAidl == null) {
203             IBatteryHAL_v2 = vendor.lenovo.hardware.battery.V2_0.IBattery.getService(); // get IBattery@2.0 service
204             IBatteryHAL_v2.linkToDeath(mHIDLDeathRecipient, 0);
205         }
206     } catch (Exception e) {
207         Slog.e(TAG, "Failed to get IBattery Service V2.0," + e);
208         try {
209             IBatteryHAL_v1 = vendor.lenovo.hardware.battery.V1_0.IBattery.getService(); // get IBattery@1.0 service
210             IBatteryHAL_v1.linkToDeath(mHIDLDeathRecipient, 0);
211         } catch (Exception e2) {
212             Slog.e(TAG, "Failed to get IBattery Service V1.0," + e2);
213         }
214     }
215     Slog.d(TAG, "mBatteryAidl = " + mBatteryAidl + ", IBatteryHAL_v2 = " + IBatteryHAL_v2 + ", IBatteryHAL_v1 = " + IBatteryHAL_v1);
216 }
217
218 private static IBinder.DeathRecipient mAidlDeathRecipient = new IBinder.DeathRecipient() {
219     @Override
220     public void binderDied() {
221         Slog.e(TAG, "IBattery aidl service Died");
222         if (mBatteryAidl != null) {
223             mBatteryAidl.asBinder().unlinkToDeath(this, 0);
224             mBatteryAidl = null;
225         }
226     }
227 };
```



05

# Question & Answer



## 参考文档

1. 硬件抽象层 (HAL) 概述 | Android 开源项目 | [Android Open Source Project \(google.cn\)](#)
2. 动态可用的 HAL | Android 开源项目 | [Android Open Source Project \(google.cn\)](#)
3. 适用于 HAL 的 AIDL | Android 开源项目 | [Android Open Source Project \(google.cn\)](#)
4. Android 接口定义语言 (AIDL) | Android 开发者 | [Android Developers \(google.cn\)](#)
5. HAL: 将 HIDL 接口改造为 Stable AIDL\_files are incompatible: the following instances ar\_fufudayo的博客-CSDN博客
6. AIDL for HALs - Code Inside [Out](#)
7. [Android Software Layer Decoupling/MSSI \(mediatek.com\)](#)
8. FCM 生命周期 | Android 开源项目 | [Android Open Source Project \(google.cn\)](#)
9. Treble | Android 开源项目 | [Android Open Source Project \(google.cn\)](#)

THANKS

**Thank You For Watching**

**Lenovo** 联想

