

Android Treble架构解析

原创

2017年12月11日 16:39:11

1955



0

本文主要介绍Treble架构下的HAL&HIDL&Binder相关技术原理。Treble的详细资料文档，请参考[Treble 官方文档](#)。



1. Treble 简介



Android 8.0 版本的新元素是 Project Treble。这是 Android 操作系统框架在架构方面的一项重大改变，旨在让制造商以更低的成本更轻松、更快速地将设备更新到新版 Android 系统。Project Treble 适用于搭载 Android 8.0 及后续版本的所有新设备（这种新的架构已经在 Pixel 手机的开发者预览版中投入使用）。



1.1 系统更新

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！



xiaosayidao

原创

8

粉丝

5

喜欢

1

评论

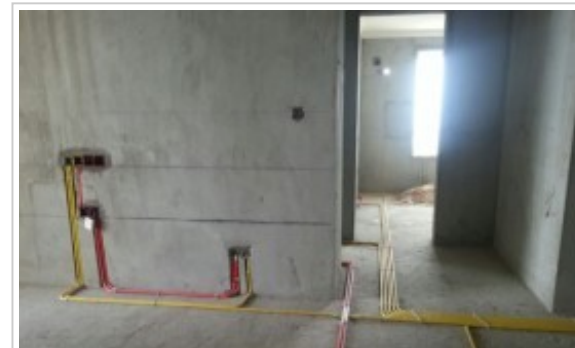
5

等级：博客 2

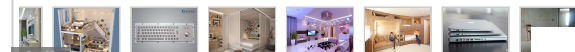
访问量：6296

积分：143

排名：113万+



装修房子顺序



广告

他的最新文章

更多

登录

注册

ANDROID UPDATES BEFORE TREBLE



图 1. Treble 推出前 Android 更新环境

Android 7.x 及更早版本中没有正式的供应商接口，因此设备制造商必须更新大量 Android 代码才能将设备更新到新版 Android 系统：

ANDROID UPDATES WITH TREBLE

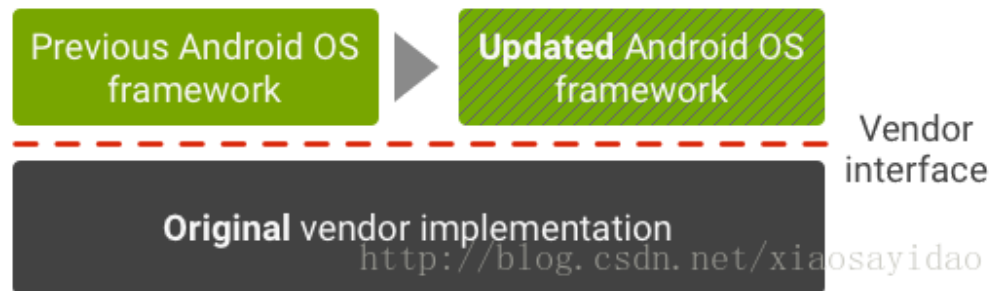


图 2. Treble 推出后的 Android 更新环境

Android aapt 生成R.java和package.apk原理解析

Android 系统服务之 ContentService

Android Apk 编译原理解析

Android 多窗口框架全解析

文章分类

android-app	1篇
Android-framework	6篇
tools	1篇

文章存档

2017年12月	2篇
2017年8月	1篇
2017年7月	3篇
2017年6月	1篇
2016年1月	1篇

他的热门文章

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

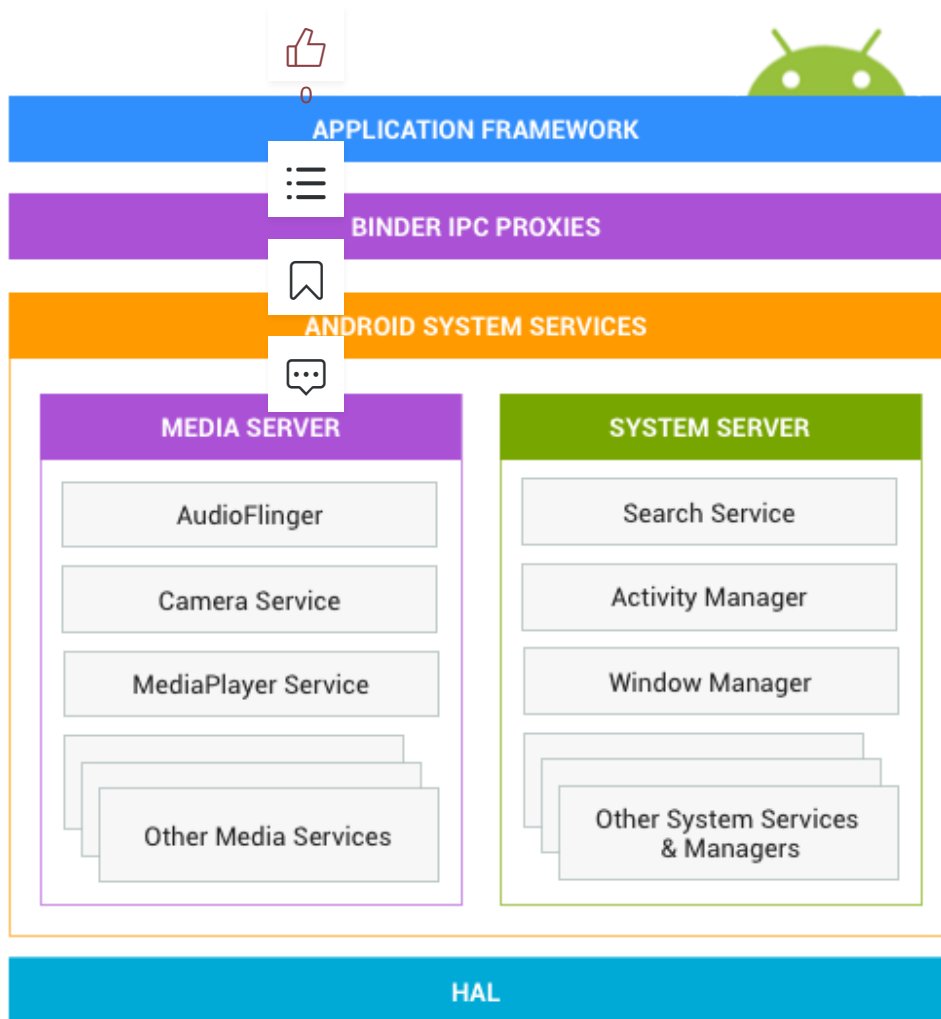
登录

注册

版本：

1.2 Android 经典架构

为了更好的了解Treble 架构里面的HAL，首先了解一下Android的经典架构。



Android 多窗口框架全解析

📖 1905

Android Apk 编译原理解析

📖 708

Android aapt 生成R.java和package.apk原理解析

📖 605

Android 系统服务之 ContentService

📖 547

Android Native Looper机制

📖 285

RuntimePermisson介绍

📖 126

Windows模拟linux终端工具Cmder+Gow

📖 94



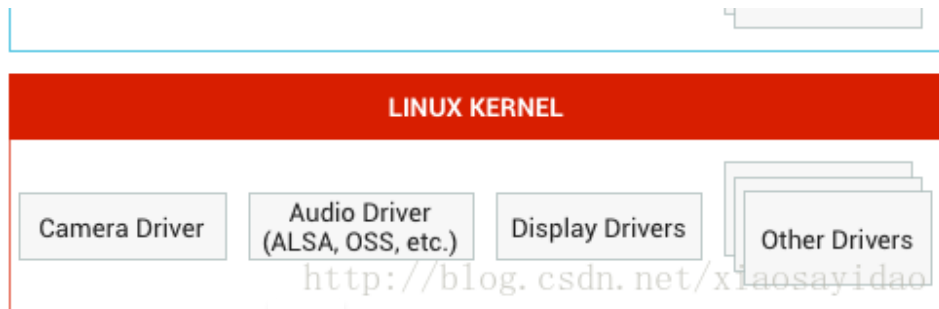
装修房子顺序



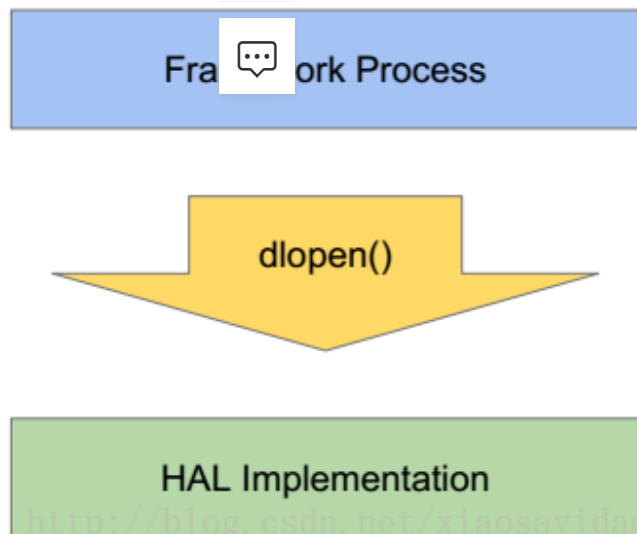
加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



在Android O之前，HAL是一个个的.so库，通过dlopen来进行打开，库和framework位于同一个进程。如图所示：



1.3 Treble 架构

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

联系我们



请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

🗨 QQ客服 🗨 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

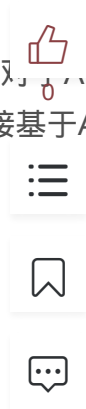
中国互联网举报中心

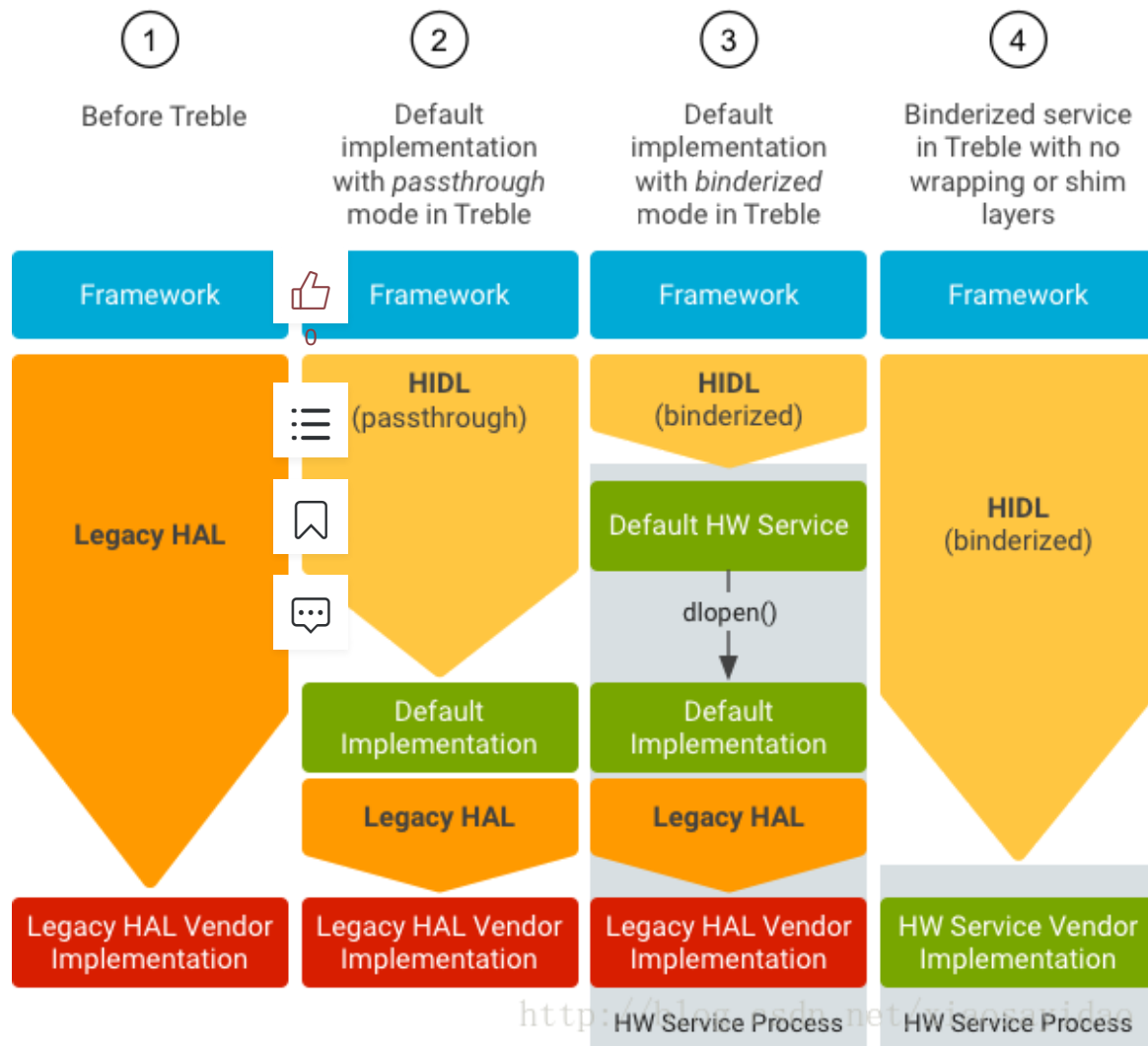
北京互联网违法和不良信息举报中心

为了能够让Android O之前的版本升级到Android O，Android设计了Passthrough模式，经过转换，可以方便的使用已经存在代码，不需要重新编写相关的HAL。HIDL分为两种模式：Passthrough和Binderized。

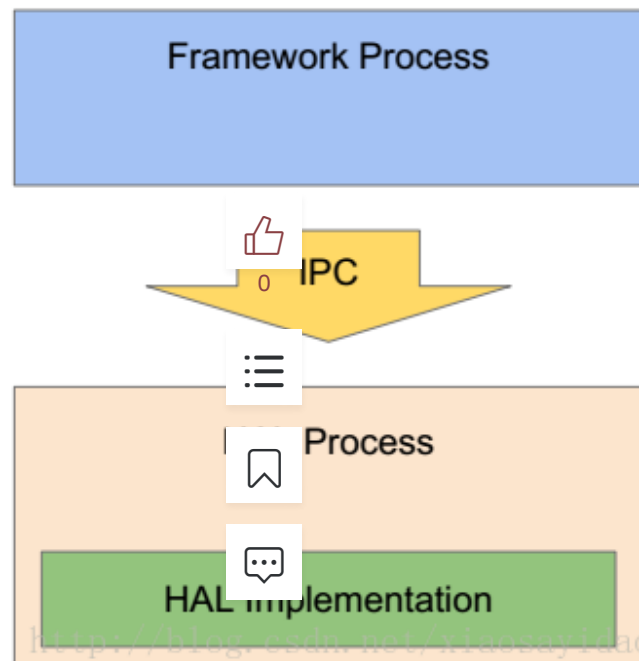
- Binderized：Google官方翻译成绑定式HAL。
- Passthrough：Google官方翻译成直通式HAL。

大致框架图如下，对于Android O之前的设备，对应图1，对于从之前的设备升级到O的版本，对应图2、图3。对于直接基于Android O开发的设备，对应图4。





新的架构之下，framework和hal运行于不同的进程，所有的HAL采用新的HIDL技术来完成。



2. HIDL 深入理解

HIDL是一种接口定义语言，描述了HAL和它的用户之间的接口。接下来深入分析一下HIDL相关实现。

2.1 hidl-gen工具

在Treble架构中，经常会提到HIDL，首先介绍和HIDL相关的一个工具hidl-gen,系统定义的所有的.hal接口，都是通过hidl-gen工具转换成对应的代码。比如hardware/interfaces/power/1.0/IP

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

0/android.hardware.power@1.0_genc++/gen/android/hardware/power/1.0/PowerAll.cpp 文件，为了深入了解，介绍相关原理,首先分析hidl-gen。

hidl-gen源码路径：system/tools/hidl，是在ubuntu上可执行的二进制文件。

使用方法：hidl-gen -o output-path -L language (-r interface-root) fqname

例子：

```
1 hidl-gen -Lmakefile -r android.hardware:hardware/interfaces -r android
```

参数含义：

- -L：语言类型，包括c++，c++-headers，c++-sources，export-header，c++-impl，java，java-constants，vts，makefile，androidbp，androidbp-impl，hash等。hidl-gen可根据传入的语言类型产生不同的文件。
- fqname：完全限定名称的输入文件。比如本例中android.hardware.power@1.0，要求在源码目录下必须有hardware/interfaces/power/1.0/目录。
 - 对于单个文件来说，格式如下：package@version::fileName，比如android.hardware.power@1.0::types.Feature。
 - 对于目录来说。格式如下package@version，比如android.hardware.power@1.0。
- -r：格式package:path，可选，对fqname对应的文件来说，用来指定包名和文件所在的目录到Android系统源码根目录的路径。如果没有制定，前缀默认是：android.hardware，目录是Android源码的根目录。
- -o：存放hidl-gen产生的中间文件的路径。我们查看hardware/interfaces/power/1.0/Android.bp，可以看到，-o参数都是写的\$(genDir)，一般都是在out/soong/.intermediates/hardware/

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

`re.power@1.0_genc++/gen`，如果是 `c++-headers`，那么就是 `out/soong/.intermediates/hardware/interfaces/power/1.0/android.hardware.power@1.0_genc++_headers/gen`。

对于实例来说，fqname是：`android.hardware.power@1.0`，包名是 `android.hardware`，文件所在的目录是 `hardware/interfaces`。例子中的命令会在 `out/soong/.intermediates/hardware/interfaces/power/1.0/` 下面产生对应的c++文件。

2.2 生成子hal **Android.mk**和**Android.bp**文件

正如我们所知，所有 **IDL Interface** 都是通过一个 `.hal` 文件来描述，为了方便编译生成每一个子hal。Google在系统提供了一个脚本 `update-makefiles.sh`，位于 `hardware/interfaces/`、`frameworks/hardware/interfaces/`、`system/hardware/interfaces/`、`system/libhidl/`。以 `hardware/interfaces` 下面的代码为实例做介绍。

```
1  #!/bin/bash
2
3  source system/tools/hidl/update-makefiles-helper.sh
4
5  do_makefiles_update \
6    "android.hardware:hardware/interfaces" \
7    "android.hidl:system/libhidl/transport"
```

这个脚本的主要作用：根据hal文件生成 **Android.mk**(makefile) 和 **Android.bp**(blueprint) 文件。在 `hardware/interfaces` 的子目录里面，存在hal文件的目录，都会产生 **Android.bp** 和 **Android.mk** 文件。详细分析如下：

a. source `system/tools` 下面的 `update-makefiles-helper.sh`，然后执行 `do_makefiles_update`

b. 解析传入进去的参数 参数 `android.hardware:hardware/interfaces`

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录


注册

- hardware/interfaces：表示相对于根目录的文件路径。

会输出如下LOG：

Updating makefiles for android.hardware in hardware/interfaces.

Updating

c. 获取所有的包名。 `function get_packages()` 函数，获取 `hardware/interfaces` 路径下面的所有 `hal` 文件所在的目录路径，比如子目录 `power` 里面的 `hal` 文件的路径是 `power/1.0`，加上当前的参数包名 `hardware/interfaces`，通过点的方式连接，将 `nfc/1.0+hardware/interfaces` 里面的斜线转换成点，最终获取的包名是 `android.hardware.power@1.0`，依次类推获取所有的包名。

d. 执行 `hidl-gen` 命令  步骤里面获取的参数和包名还有类名传入 `hidl-gen` 命令，在 `hardware/interfaces/power/1.0` 目录下产生 `Android.mk` 和 `Android.bp` 文件。

- `Android.mk`: `hidl-gen -Lmakefile -r android.hardware:hardware/interfaces -r android.hidl:system/libhidl/transport android.hardware.power@1.0`
- `Android.bp`: `hidl-gen -Landroidbp -r android.hardware:hardware/interfaces -r android.hidl:system/libhidl/transport android.hardware.power@1.0`

关于 `hidl-gen`，后续章节会介绍。

e. 在 `hardware/interfaces` 的每个子目录下面产生 `Android.bp` 文件，文件内容主要是 `subdirs` 的初始化，存放当前目录需要包含的子目录。比如 `hardware/interfaces/power/` 下面的 `Android.bp` 文件。

@hardware/interfaces/power/Android.bp

```
1 // This is an autogenerated file do not edit
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```

4     "1.0/default",
5     "1.0/vts/functional",
6 ]

```

意思就是说，编译的时候，需要编译 `hardware/interfaces/power` 目录下面的三个子目录。

经过以上步骤，就会在对应的子目录产生 `Android.mk` 和 `Android.bp` 文件。这样以后我们就可以执行正常的编译命令了。比如 `mmm hardware/interfaces/power/`，默认情况下，在源码中，`Android.mk` 和 `Android.bp` 文件已经存在。

2.3 转换.hal 为代码

如前面所示，每个接口是定义在.hal文件里面，比如 `hardware/interfaces/power/1.0/IPower.hal`，通过 `hidl-gen` 生成的 `android.bp` 文件里面会定义

```

1  filegroup {
2      name: "android.hardware.power@1.0_hal",
3      srcs: [
4          "types.hal",
5          "IPower.hal",
6      ],
7  }
8
9  genrule {
10     name: "android.hardware.power@1.0_genc++",
11     tools: ["hidl-gen"],
12     cmd: "$(location hidl-gen) -o $(genDir) -Lc++-sources -randroid.hardwar
13     srcs: [
14         ":android.hardware.power@1.0_hal",
15     ]

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```

18         "android/hardware/power/1.0/PowerAll.cpp",
19     ],
20 }

```

可以看到在 `Android.bp` 里面，通过 `hidl-gen` 在 `out` 下面产生了 `types.cpp` 和 `PowerAll.cpp`。实际例子很多，不做详细介绍。

对于生成的 `PowerAll.cpp` 来说，我们可以看到，除了 `IPower.hal` 里面定义的函数之外，还生成了很多其他的方法，这个是 `hidl-gen` 默认产生，为了能够支持 `binder` 通信。在 `IPower.hal` 里面定义的 `setInteractive(bool interactive)`，在 `PowerAll.cpp` 里面对应的是 `BpHwPower::setInteractive(bool interactive)`。通过命名就可以知道，这个和 `Binder` 机制里面的命名一致。代码如下：

```

1  ::android::hardware::Return<void> BpHwPower::setInteractive(bool interactive) {
2      atrace_in(ATRACE_TAG_HAL, "HIDL::IPower::setInteractive::client");
3      #ifdef __ANDROID_DEBUGGABLE__
4          if (UNLIKELY(mEnableInstrumentation)) {
5              std::vector<void*> _hidl_args;
6              _hidl_args.push_back((void*)&interactive);
7              for (const auto &callback: mInstrumentationCallbacks) {
8                  callback(InstrumentationEvent::CLIENT_API_ENTRY, "android.hardware.power");
9              }
10         }
11     #endif // __ANDROID_DEBUGGABLE__
12
13     ::android::hardware::Parcel _hidl_data;
14     ::android::hardware::Parcel _hidl_reply;
15     ::android::status_t _hidl_err;
16     ::android::hardware::Status _hidl_status;
17

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

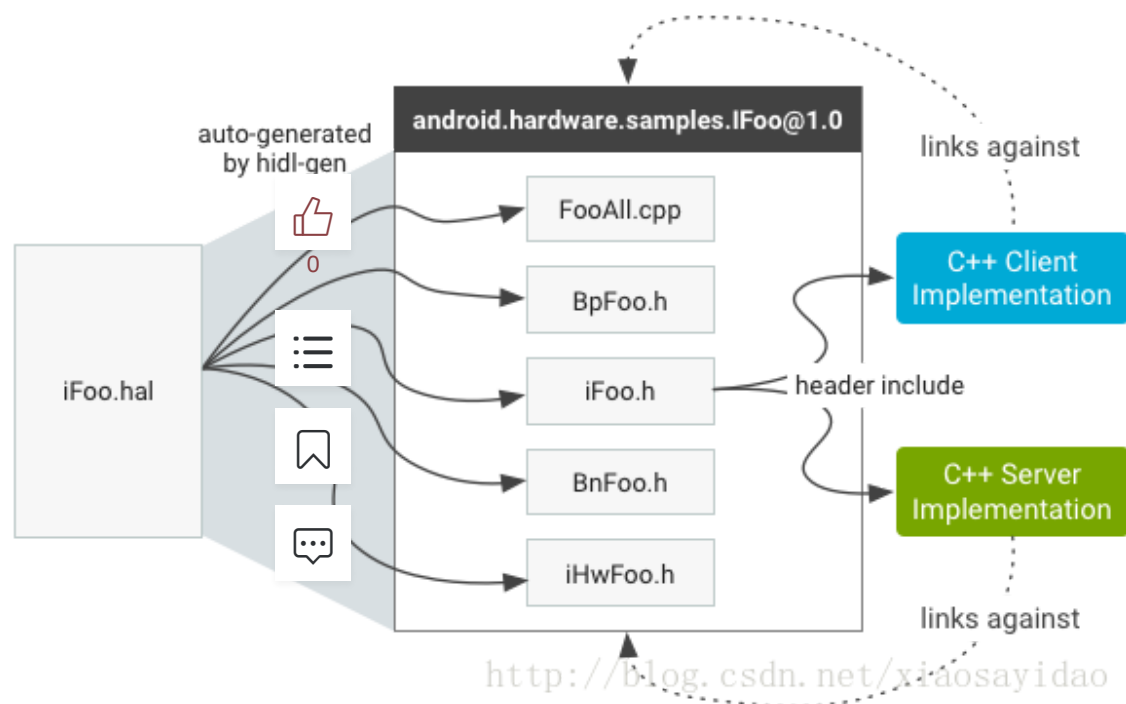
注册

```

20
21     _hidl_err = _hidl_data.writeBool(interactive);
22     if (_hidl_err != ::android::OK) { goto _hidl_error; }
23
24     _hidl_err = remote()->transact(1 /* setInteractive */, _hidl_data, &_hi
25     if (_hidl_err != ::android::OK) { goto _hidl_error; }
26
27     _hidl_ = ::android::hardware::readFromParcel(&_hidl_status, _hidl_re
28     if (_hidl_err != ::android::OK) { goto _hidl_error; }
29
30     if (!_status.isOk()) { return _hidl_status; }
31
32     atrace(ATRACE_TAG_HAL);
33     #ifdef __ANDROID_DEBUGGABLE__
34     if (UMTKFLY(mEnableInstrumentation)) {
35         std::vector<void*> _hidl_args;
36         for (const auto &callback: mInstrumentationCallbacks) {
37             callback(InstrumentationEvent::CLIENT_API_EXIT, "android.hardwa
38         }
39     }
40     #endif // __ANDROID_DEBUGGABLE__
41
42     _hidl_status.setFromStatusT(_hidl_err);
43     return ::android::hardware::Return<void>();
44
45 _hidl_error:
46     _hidl_status.setFromStatusT(_hidl_err);
47     return ::android::hardware::Return<void>(_hidl_status);
48 }

```

HIDL整个流程如图所示：



3. HAL通信机制(c++)

在Treble架构中，framework/vendor之间的通信通过 **HIDL** 接口和 **dev/hwbinder** 的IPC域来完成。而且HIDL接口有两种通信模式 **Passthrough** 和 **Binderized**。接下来我们介绍两种模式下的交互原理。创建HAL服务器有两种模式：

- defaultPassthroughServiceImplementation

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```
3 }
```

- registerAsService

```
1 int main(int /* argc */, char* /* argv */ []) {
2     sp<IDumpstateDevice> dumpstate = new DumpstateDevice;
3     config pcThreadpool(1, true /* will join */);
4     if (dumpstate->registerAsService() != OK) {
5         ALOGE("Could not register service.");
6         return 1;
7     }
8     joinRpcThreadpool();
9
10    ALOGE("Service exited!");
11    return 0;
12 }
```

接下来我们分别介绍两种类型的详细过程。

3.1 defaultPassthroughServiceImplementation

首先介绍Passthrough模式的HIDL实现机制。以 `hardware/interfaces/power/1.0` 作为例子。当编译 `hardware/interfaces/power/1.0` 的时候，会生成：

- 中间文件 `PowerAll.cpp`
- `/vendor/bin/hw/android.hardware.power@1.0-service` 的可执行文件
- `/vendor/lib/hw/android.hardware.power@1.0-impl.so` 的库文件
- `android.hardware.power@1.0-service.rc` 会被拷贝到 `vendor.img` 里面的 `vendor/etc/init` 目录

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)
[注册](#)

```

1  service power-hal-1.0 /vendor/bin/hw/android.hardware.power@1.0-service
2      class hal
3      user system
4      group system

```

接下来我们就一步步分析，power Server是如何初始化的。

- 对于init的解析  本文不做描述，在开机过程的某一个阶段，系统会启动class是hal的服务，会执行 `/vendor/bin/hw/android.hardware.power@1.0-service`，从而调用 `hardware/interfaces/power@1.0/default/service.cpp` 的 `main` 方法。代码如下：

```

1  int main()
2      return defaultPassthroughServiceImplementation<IPower>();
3  }

```

接下来会调用

@PowerAll.cpp

```

1  :android::sp<IPower> IPower::getService(const std::string &serviceName, con
2      using ::android::hardware::defaultServiceManager;
3      using ::android::hardware::details::waitForHwService;
4      using ::android::hardware::getPassthroughServiceManager;
5      using ::android::hardware::Return;
6      using ::android::sp;
7      using Transport = ::android::hidl::manager::V1_0::IServiceManager::Tran
8
9      sp<IPower> iface = nullptr;
10     // 获取HwServiceManager

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册


```

13     ALOGE("getService: defaultManager() is null");
14     return nullptr;
15 }
16 // 获取当前Transport类型, passthrough或者binderized
17 Return<Transport> transportRet = sm->getTransport(IPower::descriptor, s
18
19 if (!transportRet.isOk()) {
20     ALOGE("getService: defaultManager()->getTransport returns %s",
21         transportRet.getDescription());
22 }
23 Transport transport = transportRet;
24 const vintfHwbinder = (transport == Transport::HWBINDER);
25 const vintfPassthru = (transport == Transport::PASSTHROUGH);
26
27 // 返回当前的接口类
28
29 for (int tries = 0; !getStub && (vintfHwbinder || (vintfLegacy && tries
30     if (tries > 1) {
31         ALOGI("getService: Will do try %d for %s/%s in 1s...", tries, I
32         sleep(1);
33     }
34     if (vintfHwbinder && tries > 0) {
35         waitForHwService(IPower::descriptor, serviceName);
36     }
37     Return<sp<::android::hidl::base::V1_0::IBase>> ret =
38         sm->get(IPower::descriptor, serviceName);
39     if (!ret.isOk()) {
40         ALOGE("IPower: defaultManager()->get returns %s", ret.de
41         break;
42     }
43     sp<::android::hidl::base::V1_0::IBase> base = ret;

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```

46         ALOGW("IPower: found null hwbinder interface");
47     }continue;
48 }
49     Return<sp<IPower>> castRet = IPower::castFrom(base, true /* emitErr
50 // ...
51     iface = castRet;
52     if (iface == nullptr) {
53         ALOGW("IPower: received incompatible service; bug in hwservice
54         break;
55     }
56     return iface;
57 }
58 // 获取 Passthrough模式的类。
59 if (getBinder() == vintfPassthru || vintfLegacy) {
60     const sp<::android::hidl::manager::V1_0::IServiceManager> pm = getP
61     if (pm != nullptr) {
62         return<sp<::android::hidl::base::V1_0::IBase>> ret =
63             pm->get(IPower::descriptor, serviceName);
64         if (ret.isOk()) {
65             sp<::android::hidl::base::V1_0::IBase> baseInterface = ret;
66             if (baseInterface != nullptr) {
67                 iface = new BsPower(IPower::castFrom(baseInterface));
68             }
69         }
70     }
71 }
72 return iface;
73 }

```

• defaultPassthroughServiceImplementation() @hardware/interfaces/power/1.0/default/service

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

- `IPower::getService @PowerAll.cpp` 从`HwServiceManager`里面获取注册的服务。默认情况下是没有注册这个服务的。
- `defaultServiceManager @system/libhidl/transport/ServiceManagement.cpp` 打开 `/dev/hwbinder` , 通过binder通信, 获取`HwServiceManager`服务端。
- `sm->getTransport` 基本就是按照Binder通信的机制来实现相关的流程。通过 `HwBinder` 调用服务端的 `getTransport` 方法。



- `BpHwServiceManager::getTransport @ServiceManagerAll.cpp`
- `BpHwBinder::transact`
- `IPCThreadState::self()->transact`
- `IPCThreadState::transact writeTransactionData waitForResponse`
- `IPCThreadState::executeCommand`
- `ServiceManager::getTransport@system/hw servicemanager/ServiceManager.cpp`
 - `getTransport @ system/hw servicemanager/Vintf.cpp` 根据framework hal和device hal配置的manifest.xml里面的定义, 来判断当前的传输类型是HwBinder还是Passthrough模式。在 `vendor/manifest.xml` 里面, power配置的是hwbinder,所以最终就是hwBinder模式。(后续会讲解manifest.xml的原理)

由于我们采取的是 `defaultPassthroughServiceImplementation<IPower>()`; 进行注册, 所以 `getServicePassthroughServiceManager()` 所以会走到 `const sp<::android::hidl::manager::V1_0::IServiceManager> pm = getServicePassthroughServiceManager();`

- `getServicePassthroughServiceManager @ PowerAll.cpp` 获取passthrough服务管理。
- 调用`PassthroughServiceManager`的`get(const hidl_string& fqName, const hidl_string& name)`函数 `@ServiceManagement.cpp`, 根据传入的 `fqName=(android.hardware.power@1.0::IPower)`

加入CSDN, 享受更精准的内容推荐, 与500万程序员共同成长!

登录

注册

`id.hardware.power@1.0-impl` , 接着通过 `dlopen` 载入 `/vendor/lib/hw/android.hardware.power@1.0-impl.so` , 然后通过 `dlsym` 载入 `HIDL_FETCH_IPower` 函数。代码如下 :

@hardware/interfaces/power/1.0/default/Power.cpp

```
1  IPower* HIDL_FETCH_IPower(const char* /* name */) {
2      const hw_module_t* hw_module = nullptr;
3      power_module_t* power_module = nullptr;
4      int err = hw_get_module(POWER_HARDWARE_MODULE_ID, &hw_module);
5      if (err) {
6          ALOGE("hw_get_module %s failed: %d", POWER_HARDWARE_MODULE_ID, err)
7          return nullptr;
8      }
9
10     if (!hw_module->methods || !hw_module->methods->open) {
11         power_module = reinterpret_cast<power_module_t*>(
12             const_cast<hw_module_t*>(hw_module));
13     } else {
14         err = hw_module->methods->open(
15             hw_module, POWER_HARDWARE_MODULE_ID,
16             reinterpret_cast<hw_device_t**>(&power_module));
17         if (err) {
18             ALOGE("Passthrough failed to load legacy HAL.");
19             return nullptr;
20         }
21     }
22     return new Power(power_module);
23 }
24
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

和Power有关的接口调用，最终都是通过 `power.ranchu.so` 来实现功能。

接下来会调用 `registerReference ("android.hardware.power@1.0::IPower", "default")`，接着调用 `BpHwServiceManager::registerPassthroughClient` 将 `fqName` 和服务名，注册进 `hwserviceManager` 的 `mServiceMap` 对象里面。

```

1 Return<void> BpHwServiceManager::registerPassthroughClient(const hidl_string &fqName, const
2     cc  👍 hidl_string &name) {
3     pid_t pid = IPCThreadState::self()->getCallingPid();
4     if (!mInterfaceMap.canGet(fqName, pid)) {
5         // ⚠️ guard this function with "get", because it's typically used in
6         * the getService() path, albeit for a passthrough service in this
7         // case.
8         return Void();
9     }
10 }
11 PackageInterfaceMap &ifaceMap = mServiceMap[fqName];
12 if (name.empty()) {
13     LOG(WARNING) << "registerPassthroughClient encounters empty instance name";
14     << fqName.c_str();
15     return Void();
16 }
17 HidlService *service = ifaceMap.lookup(name);
18 if (service == nullptr) {
19     auto adding = std::make_unique<HidlService>(fqName, name);
20     adding->registerPassthroughClient(pid);
21     ifaceMap.insertService(std::move(adding));
22 } else {
23     service->registerPassthroughClient(pid);
24 }

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

- 返回 `android::hidl::base::V1_0::IBase` 实例。
- new BsPower : 首先会通过interfaceChain判断当前的interface是否支持转换，然后传入包名和接口名 `"android.hardware.power@1.0"`, `"IPower"` 构造出一个 new BsPower 的实例。
- IPower::registerAsService 接下来，调用 `status_t status = service->registerAsService(name)`，首先会创建 `BnHwPower` 对象，然后将当前的service 添加进 `hw servicemanager` 里面。初始化 `BnHwPower` 过程中，`_hidl_mImpl` 实际上就是 `BsPower` 的引用。代码如下。。

```

1  BnHwPower::BnHwPower(const ::android::sp<IPower> &_hidl_impl)
2      : ::android::hidl::base::V1_0::BnHwBase(_hidl_impl, "android.hardware.power@1.0::IPower")
3      , _hidl_mImpl = _hidl_impl;
4      , mPrio = ::android::hardware::details::gServicePrioMap.get(_hidl_mImpl, PRIORITY_DEFAULT)
5      , mSchedPolicy = prio.sched_policy;
6      , mSchedPriority = prio.prio;
7  }

```

然后调用如下步骤，将当前通信加入IPC Binder的线程池进行循环。

- `android::hardware::joinRpcThreadpool` at `system/libhidl/transport/HidlTransportSupport.cpp:28` 加入RpcThreadPool。
- `android::hardware::joinBinderRpcThreadpool` at `system/libhidl/transport/HidlBinderSupport.cpp:188`
- `android::hardware::IPCThreadState::joinThreadPool` at `system/libhwbinder/IPCThreadState.cpp:497`
- `android::hardware::IPCThreadState::getAndExecuteCommand` at `system/libhwbinder/IPCThreadState.cpp:443`

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

总结，通过 `defaultPassthroughServiceImplementation` 把当前的服务注册进 `HwServiceManager`，每个服务都是一个 `HidlService`。然后就可以等待客户端的调用。

3.2 registerAsService 创建HAL

根据[Android源码网站介绍](#)，`android.hardware.dumpstate@1.0`是属于绑定式HAL。接下来我们分析 `dumpstate` 服务初始流程。代码位于：`hardware/interfaces/dumpstate/1.0/default/`，查看 `service.cpp`，如下：

```
1 int main(int argc, char* argv[]) {
2     sp<IDumpstateDevice> dumpstate = new DumpstateDevice;
3     configureRpcThreadpool(1, true /* will join */);
4     if (dumpstate->registerAsService() != OK) {
5         ALOGE("Could not register service.");
6         return 1;
7     }
8     joinRpcThreadpool();
9
10    ALOGE("Service exited!");
11    return 1;
12 }
```

- `IDumpstateDevice::registerAsService`
- `android::hardware::details::onRegistration("android.hardware.dumpstate@1.0", "IDumpstateDevice", serviceName)`
 - `tryShortenProcessName` 设置当前进程的名字，长度最多为16。`android.hardware.dumpstate@1.0-service`

- ServiceManager::add @system/hwservice/ServiceManager.cpp 注意和binder的区别。将当前的 `service` 添加进mInstanceMap。

- 收到HwBinder驱动的 BR_TRANSACTION 消息，然后执行 BHwBinder::transact
- BnHwDumpstateDevice::onTransact
- joinRpcThreadpool(); 把当前的通信加入HwBinder的线程池进行循环。



至此，registerAsService 创建HAL Service就完成了。

3.2 Binder 模式 client和服务端的交互

服务注册成功之后， 端就可以调用相关服务提供的功能。

以点击屏幕为实例， 当我们点击屏幕的时候，会调用 `com_android_server_power_PowerManagerService.cpp` 的 `android_server_PowerManagerService_userActivity` 函数，代码如下：

```

1 void android_server_PowerManagerService_userActivity(nsecs_t eventTime, int
2     // Tell the power HAL when user activity occurs.
3     gPowerHalMutex.lock();
4     if (getPowerHal()) {
5         Return<void> ret = gPowerHal->powerHint(PowerHint::INTERACTION, 0);
6         processReturn(ret, "powerHint");
7     }
8     // ...
9 }
10 }
11
12 // Check validity of current handle to the power HAL service, and call gets
13 // The caller must be holding gPowerHalMutex

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册


```

16         gPowerHal = IPower::getService();
17         if (gPowerHal != nullptr) {
18             ALOGI("Loaded power HAL service");
19         } else {
20             ALOGI("Couldn't load power HAL service");
21             gPowerHalExists = false;
22         }
23     }
24     return gPowerHal != nullptr;
25 }
26

```

在 `getPowerHal` 里通过 `IPower::getService();` 方法经过 `HwBinder` 通信，获取服务端的引用。主要包含如下步骤：

- `IPower::getService` 获取 `IPower` 的服务。返回远程服务的代理 `gPowerHal`，最终返回的是 `BpHwPower`。
- `IPower::getService(const std::string &serviceName, const bool getStub)@PowerApp.cpp`。
- `BpHwServiceManager::getTransport` 获取当前的传输类型，`passthrough` 或者 `binderized`。Power 是 `binderized`，返回对应的服务代理。
- `sm->get(IPower::descriptor, serviceName)` 从 `ServiceManager` 里面获取描述是 `android.hardware.power@1.0::IPower`，服务名是 `default` 的 `hidlservice` 的引用。
- `IPower::castFrom(base, true /* emitError */)`
- `android::hardware::details::castInterface` 将 `hidlservice` 服务的引用转换成 `Binder` 对象。
- `::android::hardware::IInterface::asBinder(static_cast`

查询manifest.xml可以发现。android.hardware.graphics.mapper是passthrough的模式。

```

1    <hal format="hidl">
2        <name>android.hardware.graphics.mapper</name>
3        <transport arch="32+64">passthrough</transport>
4        <version>2.0</version>
5        <interface
6            <name>IMapper</name>
7            <instance>default</instance>
8        </interface>
9    </hal>

```

以hardware/interfaces/graphics/mapper/2.0/作为例子进行分析。

@frameworks/native/ui/Gralloc2.cpp

```

1  Mapper::Mapper()
2  {
3      mMapper = IMapper::getService();
4      if (mMapper == nullptr || mMapper->isRemote()) {
5          LOG_ALWAYS_FATAL("gralloc-mapper must be in passthrough mode");
6      }
7  }

```

// static

```

::android::sp IMapper::getService(const std::string &serviceName, const bool getStub) {
using ::android::hardware::defaultServiceManager;
using ::android::hardware::details::waitForHwService;
using ::android::hardware::getPassthroughServiceManager;

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```

using ::android::sp;
using Transport = ::android::hidl::manager::V1_0::IServiceManager::Transport;

1  sp<IMapper> iface = nullptr;
2
3  const sp<::android::hidl::manager::V1_0::IServiceManager> sm = defaultServi
4  if (sm == nullptr) {
5      ALOGE("Service: defaultManager() is null");
6      return nullptr;
7  }
8
9  Return<Transport> transportRet = sm->getTransport(IMapper::descriptor, serv
10
11  if (!transportRet.isOk()) {
12      ALOGE("Service: defaultManager()->getTransport returns %s", t
13      return nullptr;
14  }
15  Transport transport = transportRet;
16  const bool vintfHwbinder = (transport == Transport::HWBINDER);
17  const bool vintfPassthru = (transport == Transport::PASSTHROUGH);
18
19  // ...
20  if (getStub || vintfPassthru || vintfLegacy) {
21      const sp<::android::hidl::manager::V1_0::IServiceManager> pm = getPasst
22      if (pm != nullptr) {
23          Return<sp<::android::hidl::base::V1_0::IBase>> ret =
24              pm->get(IMapper::descriptor, serviceName);
25          if (ret.isOk()) {
26              sp<::android::hidl::base::V1_0::IBase> baseInterface = ret;
27              if (baseInterface != nullptr) {

```

```

31     }
32 }
33 return iface;
34
}

```

- 步骤和前面的- 由于是passthrough的模式，调用 `PassthroughServiceManager` 的 `get(const hidl_string& qName, const hidl_string& name)` 函数 `@ServiceManagement.cpp`，根据传入的 `fqName=(android.hardware.graphics.mapper@2.0::IMapper)`，获取当前的接口名 `IMapper`，拼接 `HIDL_FETCH_IMapper` 和库名字 `android.hardware.graphics.mapper@2.0-impl`，接着通过 `dlopen` 载入 `android.hardware.graphics.mapper@2.0-impl`，然后通过 `dlsym` 载入 `HIDL_FETCH_IMapper` 函数。

这样就实现了passthrough模式下的通信了。

4. HAL 通信 (JAVA)

以 `hardware/interfaces/radio/1.0/` 作为例子：

当我们编译 `hardware/interfaces/radio/1.0/` 的时候，会编译出：

- `android.hardware.radio-V1.0-java-static`
- `out/target/common/gen/JAVA_LIBRARIES/android.hardware.radio-V1.0-java-static_intermediates/android/hardware/radio/V1_0/IRadio.java`

接下来我们以

`@frameworks/opt/telephony/Android.mk` 最为例子，直接引用 `android.hardware.radio-V1.0-jav`

```

1
2 LOCAL_PATH := $(call my-dir)
3 include $(CLEAR_VARS)
4
5 // ...
6 LOCAL_JAVA_LIBRARIES := voip-common ims-common
7 LOCAL_STATIC_JAVA_LIBRARIES := android.hardware.radio-V1.0-java-static \
8     android.hardware.radio.deprecated-V1.0-java-static
9 LOCAL_MODULE_TAGS := optional
10 LOCAL_MODULE := telephony-common
11 // ...
12
13 include $(LOCAL_PATH)/jni/D_JAVA_LIBRARY)

```

接下来我们看一下这个地方。

@RIL.java

```

1      try {
2          mRadioProxy = IRadio.getService(HIDL_SERVICE_NAME[mPhoneId == n
3          if (mRadioProxy != null) {
4              mRadioProxy.linkToDeath(mRadioProxyDeathRecipient,
5                  mRadioProxyCookie.incrementAndGet());
6              mRadioProxy.setResponseFunctions(mRadioResponse, mRadioIndi
7          } else {
8              riljLoge("getRadioProxy: mRadioProxy == null");
9          }
10     } catch (RemoteException | RuntimeException e) {
11         mRadioProxy = null;
12         riljLoge("RadioProxy getService/setResponseFunctions: " + e);

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

首先会直接调用 `IRadio.getService` 来获取相关服务。

@IRadio.java

```
1 public static IRadio getService(String serviceName) throws android.os.R
2     return IRadio.asInterface(android.os.HwBinder.getService("android.h
3 }
4
```



0

android.os.HwBinder.getService("android.hardware.radio@1.0::IRadio",serviceName)



JNI

@frameworks/base /jni/android_os_HwBinder.cpp



```
1 static jobject JHwBinder_native_getService(
2     JNIEnv *env,
3     jclass /* clazzObj */,
4     jstring ifaceNameObj,
5     jstring serviceNameObj) {
6
7     ///...
8
9     auto manager = hardware::defaultServiceManager();
10
11     // ...
12
13
14     Return<IServiceManager::Transport> transportRet =
15         manager->getTransport(ifaceNameHStr, serviceNameHStr);
16
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```

20     }
21
22     IServiceManager::Transport transport = transportRet;
23
24     // ... java 类型的传输模式必须是HwBinder
25
26     if (transport != IServiceManager::Transport::HWBINDER && !vintfLegacy)
27         LOG(ERROR) << "service " << ifaceName << " declares transport metho
28             << toString(transport) << " but framework expects hwbind
29             signalExceptionForError(env, UNKNOWN_ERROR, true /* canThrowRemoteE
30             return NULL;
31     }
32     // 获取引用。
33     Return hidl::base::V1_0::IBase>> ret = manager->get(ifaceNameHStr, s
34
35     if (!sOk()) {
36         signalExceptionForError(env, UNKNOWN_ERROR, true /* canThrowRemoteE
37         return NULL;
38     }
39
40     // 转换成Binder接口
41     sp<hardware::IBinder> service = hardware::toBinder<
42         hidl::base::V1_0::IBase, hidl::base::V1_0::BpHwBase>(ret);
43
44     if (service == NULL) {
45         signalExceptionForError(env, NAME_NOT_FOUND);
46         return NULL;
47     }
48
49     LOG(INFO) << "Starting thread pool.";
50     android::hardware::ProcessState::self()->startThreadPool();

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

```
53     return JHwRemoteBinder::NewObject(env, service);
    }
```

以上步骤和C++里面的获取服务步骤类似。通过 `IRadio.getService()` 获取相关的服务，进入JNI的相关接口，获取 `HwServiceManager` 服务，然后获取当前HAL的类型（必须是Binderized），接下来获取服务对应的接口，接着将当前接口转换成Ibinder引用，然后创建 `JHwRemoteBinder` 对象返回给java层。



```
1 IRadio.asInterface(android.os.HwBinder.getService("android.hardware.radio@1
```

java层接着调用 `IRadio.asInterface` 将 `Hwbinder` 引用转换成 `IRadio` 对象。

这样就可以通过 `IRadio` 对象调用

5. Vendor Interface Object

5.1 manifest.xml 和 compatibility_matrix.xml

在system分区和vendor分区，分别存在manifest.xml和compatibility_matrix.xml。内容大致如下：

```
1 <manifest version="1.0" type="framework">
2     <hal format="hidl">
3         <name>android.frameworks.displayservice</name>
4         <transport>hwbinder</transport>
5         <version>1.0</version>
6         <interface>
7             <name>IDisplayService</name>
```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册


```

11     <hal format="hidl">
12         <name>android.frameworks.schedulerservice</name>
13         <transport>hwbinder</transport>
14         <version>1.0</version>
15         <interface>
16             <name>ISchedulingPolicyService</name>
17             <instance>default</instance>
18         </interface>
19     </hal>
20     ...
21 </manifest>

```

分为两类：

- framework相关，Google默认定义完成。
- device相关，厂商自定义。

device可以通过 **DEVICE_MANIFEST_FILE** 和 **DEVICE_MATRIX_FILE** 指定自己的manifest.xml文件。如高通平台的项目：

```

1  DEVICE_MANIFEST_FILE := device/qcom/msm8937_64/manifest.xml
2  DEVICE_MATRIX_FILE   := device/qcom/common/compatibility_matrix.xml

```

默认的framework manifest定义和兼容性文件定义如下：

@build/core/config.mk

```

1  FRAMEWORK_MANIFEST_FILE := system/libhidl/manifest.xml
2  FRAMEWORK_COMPATIBILITY_MATRIX_FILE := hardware/interfaces/compatibility_ma

```

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

通过对比可以发现，out下面生成的和源码里面存在的文件，并不是完全一致，在Android.mk里面可以发现，这几个文件都经过了out/host/linux-x86/bin/assemble_vintf转换，assemble_vintf会判断文件格式是否正确，并且会根据name按字母顺序排列。

以上两个xml都是在，在system/libvintf/parse_string.cpp里面进行解析。

在前面的介绍中，我们都讲到了一个重要的方法，就是transport

在system/libvintf/include/vintf/Transport.h定义

```
1 static constexpr std::array<std::string, 3> gTransportStrings = {  
2     {  
3         "",  
4         "through",  
5         "der",  
6     }  
7 };
```

我们获取服务的时候，首先肯定要获取当前的HAL是什么类型。

6 其他技巧

打印当前的manifest信息

- mmm system/libvintf/
- adb push out/target/product/(产品名)/system/bin/vintf /system/bin/vintf
- adb shell vintf

版权声明：本文为博主原创文章，未经博主允许不得转载。

<http://blog.csdn.net/xiaosayidao/article/details/75577940>



0



目前您尚未登录，请 [登录](#) 或 [注册](#) 后进行评论



greatwh 2018-02-18 11:46

1条回复 回复 2楼

楼主你好，有个 就以你举例的Power类为例，代码上来看明明是passthrough的方式，但是在manifest.xml中却定义产 nder的方式，为何？

```
<hal format="hidl" 
<name>android.hardware.power</name>
<transport>hwbinder</transport>
<version>1.0</version>
<interface>
<name>IPower</name>
<instance>default</instance>
</interface>
</hal>
```



weixin_40815661 2018-01-11 16:18

回复 1楼

在网上能找到的hidl相关的最好的文章

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)

[注册](#)

Android O 前期预研之一：Android Treble 计划

Android O 前期预研之前一直传言android O的推出会包含有两部分的主要功能：谷歌打算彻底解决Android版本碎片化的问题，会在Android O上推出一个新的框架设计来解决这个这个升...



ljp1205 2017年08月29日 18:20 5549

漫谈android系统-Android O的Android Treble 计划与大的变动

Android O的大变更Android O这次来了一个大手笔，谷歌的人第一次到下游厂商进行宣导，并指出了并开始向各家厂商灌输谷歌的新理念：彻底解决Android版本碎片化的问题我有幸参加了这次宣讲，...



u013983194 2017年03月30日 22:10 936

技术外文文献看不懂？教你一个公式秒懂英语

不背单词和语法，一个公式学好英语



Android-Treble-简要介绍

2017年11月10日 09:23 3.73MB

下载



Android Treble架构解析



omnispac 2018年01月26日 09:10 235

本文主要介绍Treble架构下的HAL&HIDL&Binder相关技术原理。Treble的详细资料文档，请参考Treble 官方文档。1. Treble 符合 Android O 版本的

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

Android O 前期预研之一：Android Treble 计划

本文章转载自：<http://blog.csdn.net/ljp1205/article/details/77684550> 作者：影子LEON Android O 前期预研 之前一直传言androi...



solid_sdu 2017年11月09日 10:18 187

IT采购开年钜惠，阿里云百款产品5折起

开年采购季，阿里云新则再返最高6000



Android HIDL 官方文档（六）—— 使用 Binder 进程间通信机制（Using Bind...

1. Binder 驱动的改变 1. Binder 域（上下文） 1.2 散集列表 1.3 更细化地加锁 1.4 实时优先级继承 1.5 用户空间更改 1.6 公共内核中的一些 SHA ...



qq_16775897 2017年10月31日 20:30 2584

安卓8.0 WIFI差异分析



u010842019 2017年12月16日 17:03 1294

===== /packages/apps/Sett
ings/src...

Android O 前期预研之二：HIDL相关介绍



ljp1205 2017年09月07日 00:19 10033

在上一篇博客里，大致介绍了下Android O 中treble计划的一些背景与相关基本架构，这一篇中跟大家一起来探讨下HIDL相关的内容。Android HAL类型 在此之前的ANDROI...

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册

学习 Android O HIDL



ch853199769

2017年09月04日 17:57

6261

HIDL 简介 Project Treble 关于HIDL的设计 使用 passthrou
gh 模式 Passthrough header files Binderizing passthrough ...

Android HIDL 官方文档（一）—— 概述（Overview）

最近因为业务上的需求 让我先看着 HIDL 的官方文档学习学习。然而直接看英文文档还是很不习惯，就打算一边翻译一边学习。翻译相关的内容都是以中英文对照的形式贴出来。...



qq_16775897

20



0月21日 23:10

902

AndoridO hidl



prike

2017年11月08日 10:11

685

HIDL 简介 Project Trebl HIDL的设计使用 passthrough 模式Passthrough header filesBinderizing passthrough H...

程序员不会英语怎么行？

老司机教你一个数学公式秒懂天下英语



从CarAudioManager调用流程开始学习Vendor Interface（Service部分）

接下来就是Vendor的实现了！如果要作为一个Service来提供，我们应该怎么设计呢？TODO: QA:怎么设计比较合理。首先，在Android 系统启动的时候init进程 /system/c...



bberdong

2018年01月15日 13:45

331

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)[注册](#)

关于 Android O 的 treble/hidl



leonxu_sjtu

2018年02月12日 10:40

📖 135

一个月之前就已经做了 Android O 的预研，整理了 Android O 上 Media 部分的更新点，比如 Drm, MediaCas, MediaMetrics, PIP 等等，但是给...

Android Binder学习(三)之defaultServiceManager()的分析

Android Binder学习(三):



ultServiceManager()的分析

文章还是按着函数调用的顺序来分析的。这里我们就在

mediaServer进程中研究一下，serviceMa...



armwind

2017年



1日 09:25



📖 2759

service 命令



q1183345443

2016年10月24日 17:58

📖 400

之前在使用adb时，用过



命令 比如adb install 等最后是调用了 pm相关命令，和Java service相关的命令都在Framework/base/cmds目录下，今天我们来讲下serv...

【Bash百宝箱】从Android.mk到Android.bp

最近更新了Android Nougat源码，无意间发现Android的编译系统已经发生了巨大改变，到处是“Android.bp”文件，下面就来看一下这个bp文件到底是何方神圣。首先从Soong说起，S...



iEearth

2017年01月24日 14:01



📖 10631

android.bp



prike

2017年10月24日 10:40

📖 1627

最近更新了Android Nougat源码，无意间发现Android的编译系统已经发生了巨大改变，到处是“Android.bp”文件，下面就来看一下这个bp文件到底是何方神圣。首先从Soong说起，S...

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

[登录](#)[注册](#)

开源商城系统

谁知道开源商城的系统有哪些

百度广告

Android7.0 Ninja编译原理



ChaoY1116

2016年11月07日 06:52

16451

引言 使在Android N的系统上，初次使用了Ninja的编译系统。对于Ninja，最初的印象是用在了Chromium open source code的编译中，在chromium的编译环境中，...

Android 编译系统 Android.bp



drageon_j

2017年08月17日 17:31

1968

从Android 7.0 (N)开始, Google开始逐步使用Android.bp代替原来的Android.mk进行编译. Google称之为soong, 具体可以参考: <https://and...>

mtk android.mk --> android.bp



qq_37610155

2018年01月09日 16:14

317

mtk android.mk --> android.bp 例如\frameworks\base\media\jni 由7.0的android.mk转换成android.bp . 首先从S...

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册