

# Bitcomet设计开发文档

高校队伍：广东东软学院 Bitcomet队

项目成员：邹明燊、田梓汎、张阳彬

指导教师：刘翠莲、罗泉

## 简介

Bitcomet是一个基于原有Anbox向Android 11移植实现一种基于Linux系统运行Android 11的解决方案，以下均**称呼为Bitcomet**。其主要与原有Anbox的**改进点**在于Android所运行的环境由LXC实现变为更加主流的Docker实现，原有在Anbox不再更新的Android 7更新为Android 11。此外，为了让Android 11能在Docker中运行，本项目针对其**SDcard的文件系统挂载机制、SELinux安全机制**等进行了相应的移植与优化。为了让Android能在Linux上满足正常使用以及调用GPU进行渲染的需求，本项目**参考并移植**原有Anbox以及Google安卓模拟器中的相关实现，在移植并搭建起Anbox的跨系统环境通信基础上，使相应的**Android各类接口打通到外部Linux**，其中最基础的包括**图形渲染、图形输出、键鼠输入**这三个部分的接口。

本项目主要还是基于Anbox来参考实现，旨在完成以下四个目标：

- **目标1：**完成Android11的**SDcard的文件系统挂载机制、SELinux安全机制、非特权模式下运行处理**对应在Docker容器环境内的处理。
- **目标2：**完成初步目调试与测试的Demo1版本，使用adb配合Scrcpy对内部Android远程访问以方便有个初步预期，并且方便本项目有个初步测试的环境。
- **目标3：**完成Anbox通信部分实现向Android 11的移植，使Android能与Anbox外部实现跨系统环境通信。
- **目标4：**完成Anbox安卓其余部分的各个模块移植，使Android的三个基础部分OpenGL、HWC图形输出、键鼠输入能正常工作。

目前，我们的赛题完成度如下：

表1.1 赛题完成度

目标编号	基本完成情况	额外说明
1	基本完成 (≈85%)	1. 与SDcard以及SELinux组件相关的Android 11的基本功能、SDcard文件管理、安装第三方软件、闹铃与联系人等基础应用测试均通过。 2. 非特权模式仅做了部分处理，未进行测试。
2	基本完成 (≈90%)	1. 搭建好相关容器，进行相关基础测试并通过。 2. 目前网络部分由于在Docker特权模式下运行，可能会有Bug。

目标编号	基本完成情况	额外说明
3	初步完成 (≈80%)	<p>1. 完成跨系统环境通信，安卓中Anbox的相关模块可以使用QEMU_PIPE（实际容器中是没有QEMU设备，而是使用了Unix Domain Socket代替，但其通信的相关通道仍然叫做QEMU PIPE）进行通信或者使用Anbox实现的RPC调用。</p> <p>2. Anbox的OpenGL ES的emulation库实现初步测试能通过该通信创建QEMU_PIPE连接，实现RPC调用获取到主机侧支持的OpenGL信息。</p> <p>3. 由于各个部分的模块暂未完全移植好，还需要全部移植才能测试所有功能。</p>
4	初步测试 (≈60%)	<p>1. 已经把相应的Anbox的各个模块的移植到Android中，其中GPS与Audio模块使用了谷歌给Goldfish的实现。</p> <p>2. OpenGL ES实现和HWC实现在Android 11下无法测试，由于Android 11的相关变动，导致输出画面调用了Gralloc的PostFB去输出，而在Anbox中是未做这部分RPC调用的，输出画面失败。</p> <p>3. 把相关方案在Android 10上进行测试，测试可以输出画面，但卡在安卓Launcher第一屏画面。</p>
总计	≈80%	还有更多的工作需要测试和完成

## 目录

### Bitcomet设计开发文档

简介

目录

1. 概述

1.1 Bitcomet方案设计与分析

1.1.1 为什么选择使用容器与Anbox

1.1.2 总结

1.2 项目的主要工作

2. Bitcomet方案设计

2.1 Docker下的安卓11容器设计

2.1.1 主要工作

2.1.2 Bitcomet底层部分特点

2.2 安卓与上层Anbox通信设计

2.2.1 主要工作

2.2.2 Bitcomet中间层部分特点

2.3 上层Anbox的OpenGL ES渲染以及输入输出设计

2.3.1 主要工作

2.3.2 Bitcomet上层部分特点

3. Bitcomet方案测试

4. 未来展望

5. 参考文献

# 1. 概述

## 1.1 Bitcomet方案设计与分析

### 1.1.1 为什么选择使用容器与Anbox

本Bitcomet方案旨在于实现在Linux中运行Android 11的解决方案。在这个方案中会遇到一些问题，如下图1.1是Android系统框架图、图1.2GNU/Linux系统架构图，这两张图清晰明了地展现了Android与Linux环境的差异：

Android与Linux除了内核相似外是**完全不一样的系统环境**，尤其体现在系统libc库、应用的运行方式、输入输出设备的管理方式、图像合成方式、OpenGL接口、网络管理组件等方面。这些组件绝大部分将会与Linux**冲突**，导致Linux运行异常或者Android无法直接运行，同时Android的APP也无法在Linux内直接运行。

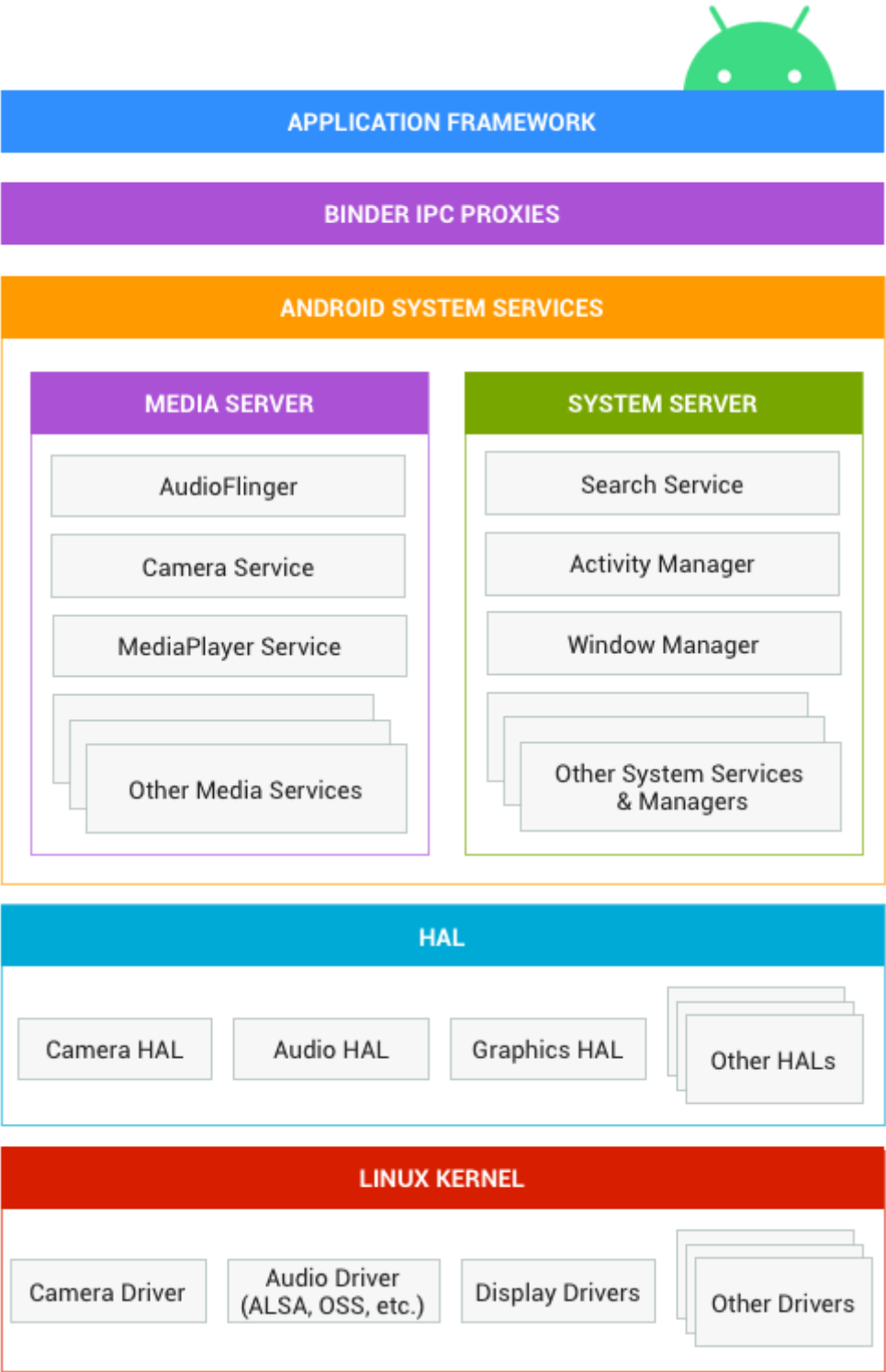


图1.1 Android 系统架构图（图源自参考文献2）

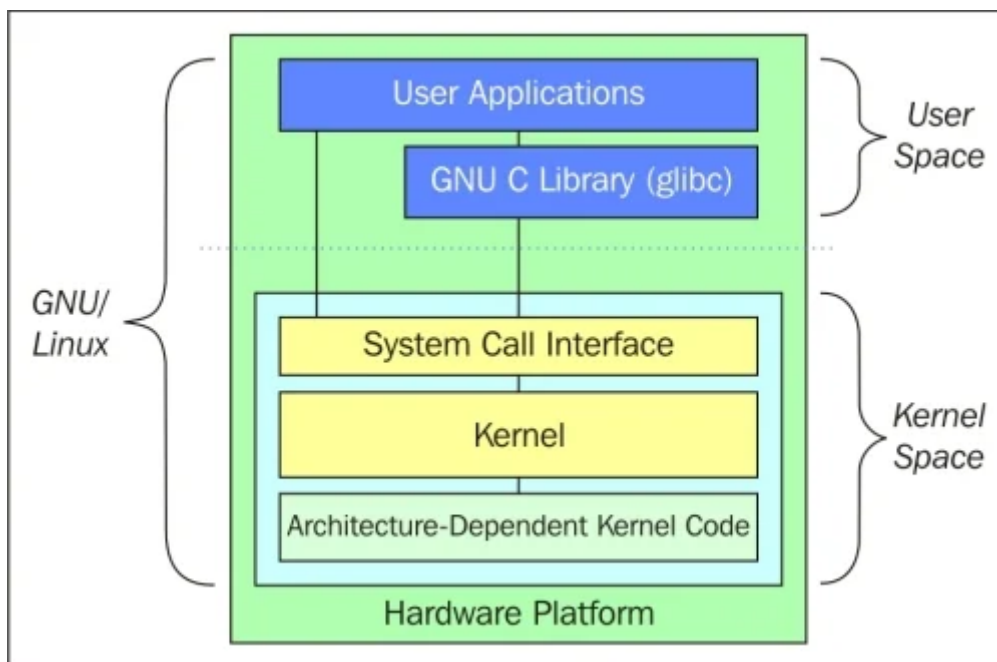


图1.2 GNU/Linux系统架构图（图源自<https://zhuanlan.zhihu.com/p/43029021>）

**旧方案：**在过去面向用户的**常见方案**中，常用虚拟机来实现这个方案。利用**虚拟化**功能运行一个Android虚拟机，虚拟机内虚拟或者模拟了电脑绝大部分的硬件，这样既能解决两者接口不同，又能简单实现Android在其他系统上的运行。但是随之而来也有许多**缺点**：虚拟化需要硬件支持、虚拟化对CPU性能有一小部分损耗、虚拟机本身运行一个全新的系统也需要更多硬件资源、在虚拟机与主机系统间切换也无法很好地利用上常见PC系统的多任务多窗口特性。

相比起虚拟机的实现方法，容器基于Linux内核的命名空间隔离，也能提供一个**隔离**了Android系统与Linux之间的磁盘、内存等的运行环境。同时容器**不需要虚拟化，不需要跑起整个Linux内核**，相对应的对硬件资源要求较少，这三个方面看起来有优势。但是容器又出现不一样的问题：相比一般Android模拟器使用的虚拟机，容器又缺少输入输出接口、3D加速与显示输出。

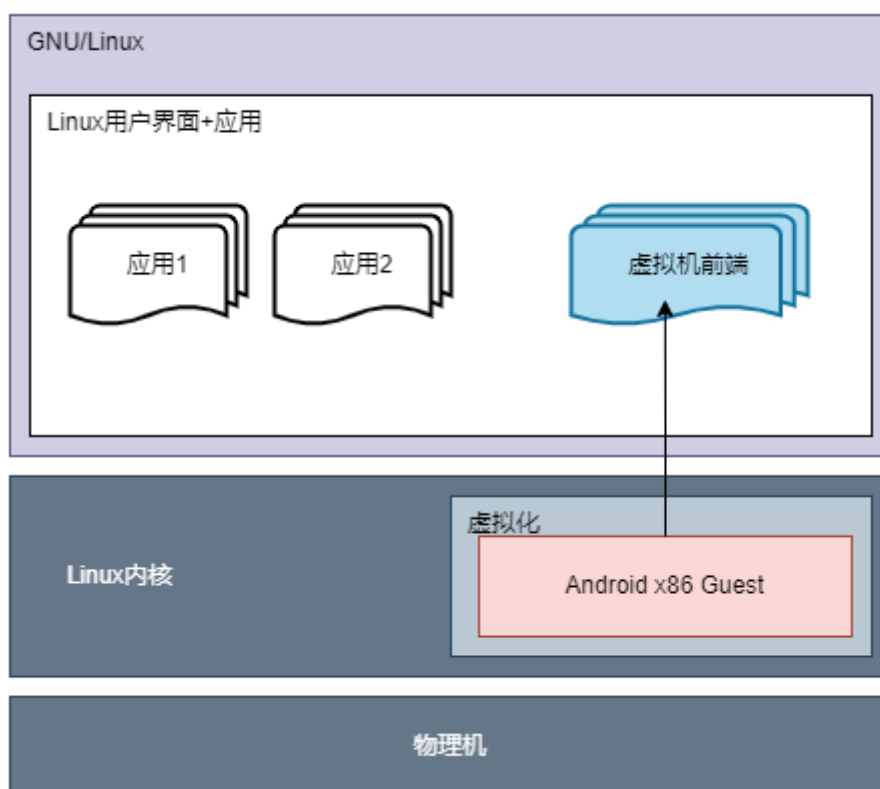


图1.3 虚拟机运行Android x86架构图

**Bitcomet方案：**使用容器提供Android运行的环境来实现，其原因如下：该方案可能会在国产化平台上运行，面向不同的硬件环境下不一定支持虚拟化；对硬件资源需求较少；针对容器缺少的输入输出接口以及3D加速与显示输出，谷歌为解决这些问题在安卓模拟器AVD上设计了一套新的方案。另外Anbox在一部分输入输出及3D加速方面正好是复用了谷歌的安卓模拟器的方案，利用了其提供的QEMU\_PIPE、emugl、传感器模拟、音频输出等方面提供的实现。

以上是复用了谷歌的实现，但是前端不一样，安卓模拟器的前端是方便谷歌模拟器实现的，而Anbox是需要利用上PC系统的**多任务多窗口**、方便对接输入输出和控制等的特性。因此Anbox这部分针对谷歌的方案进行了**参考重写**，把Android中各个不同APP的窗口映射到不同的模拟Surface上。但相应的代价是，由于谷歌3D渲染部分是通过把其指令传输出来交给**外部**OpenGL渲染，这样性能利用是真机渲染的70%-90%（数据来自参考文献[1]），有一定的损耗。

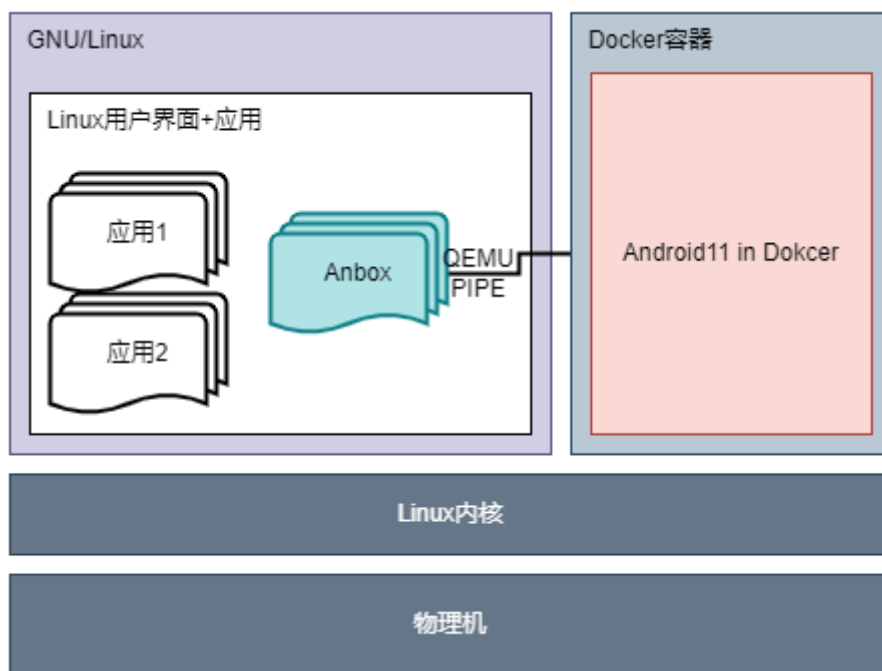


图1.4 Docker运行Android架构图

### 1.1.2 总结

本项目是这类移植方式中其中一种兼容性较好的方案，其OpenGL ES渲染是通过**传输**给Anbox并由emugl转换后交给Linux**用户空间**的OpenGL库实现，OpenGL的对应版本的API是**统一**的，而Linux平台一般都有GPU驱动+相应库，即使没有也会使用软件OpenGL库，所以这种方案理论上是能**适应**更多GPU平台的。

总的来说这类移植方式主要都有一类**共通点**，为了把输入输出、显示渲染等**接口打通到对应平台**，而不断衍生的技术都有新的优化与改进，尽量减少通信期间数据的拷贝，降低损耗，提高硬件利用率。

## 1.2 项目的主要工作

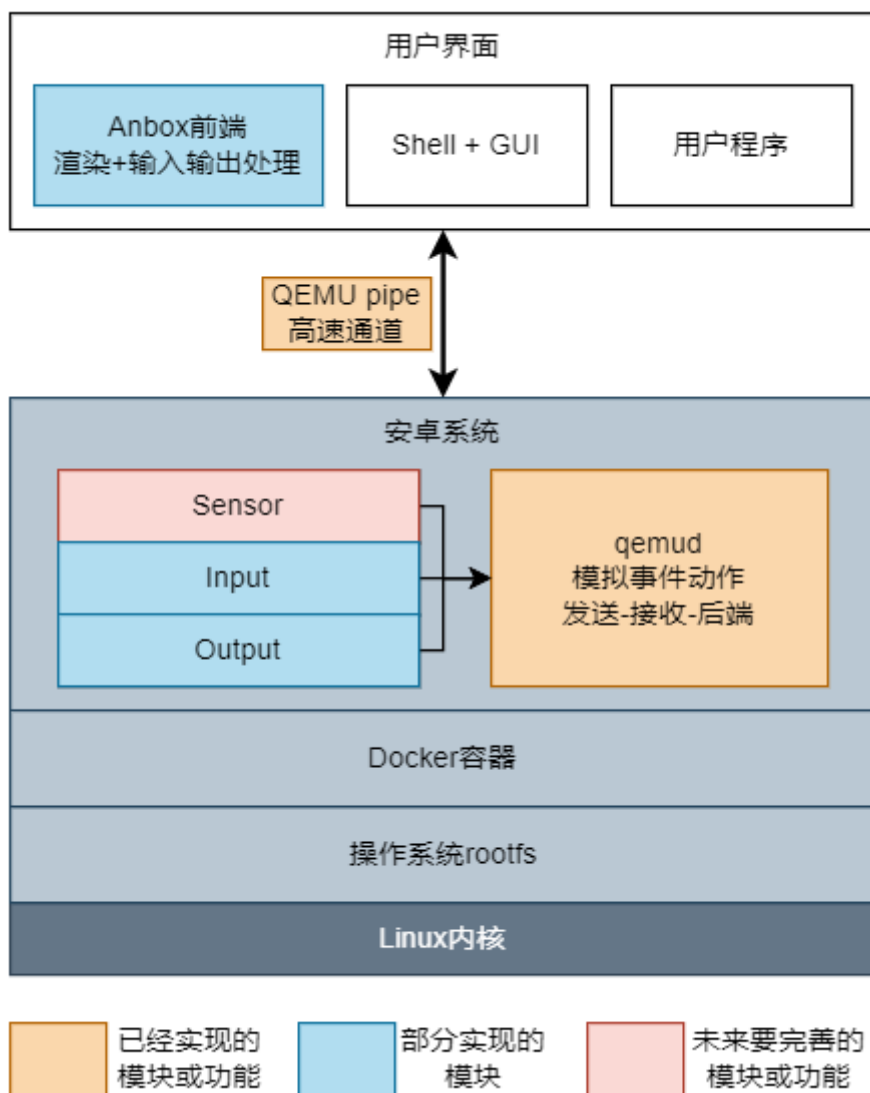


图1.6 Bitcomet解决方案架构图

该种方案从结构上来说适合其他类似移植方案的参考，例如移植其他系统到Linux中时，其图形与输入输出部分的实现可参考本方案的思路。本方案的主要工作如下：

此项目基于原有Anbox方案进行**更新及改进**。我们**改进**使用基于Docker容器技术提供隔离的、与Linux并行运行的安卓环境，此种改进是对原有LXC容器的**更优替代**。但是该改进需要重新适配与调试，需要大量移植与测试的工作量才能保证维持原有稳定性，并更好地利用更多的新特性开发新的功能。除此之外**更新**新的Android 11替代原有Android 7，此过程需要重新移植原有Anbox的各个组件适配新安卓上的HIDL接口。同时需要修改Android底层，经过大量的移植和测试的工作，让其能正常运行在Docker容器中。此项目利用Anbox部分现成的模块，在Linux下上层APP实现与安卓底层对接输入输出和OpenGL ES的渲染等实现。这些设计目标在于使用Docker**替代**原有Anbox使用的较旧的LXC容器技术，并**更新**其Android 7到Android 11，其中技术革新点如下：

- (1). 使用更新的Android 11替代原有老旧的**Android 7**。
- (2). Android 11中的HAL层使用了HIDL接口这一新的**Treble**的架构，更加规范及更高的可移植性。
- (3). 使用生态丰富、快速部署、可移植、更加可控的**Docker**作为底层容器运行环境替代原有集成LXC这一高耦合、低可移植性的实现方式。
- (4). 活用**成熟**的Anbox部分对多平台高兼容性的实现方式，以应对目前国产系统多种硬件平台和系统环境的问题。

## 2. Bitcomet方案设计

本方案是在Ubuntu 20.04(下图GNU/Linux部分)，部署好Docker环境。

**Bitcomet底层：**其中Android 11的Docker镜像以及Anbox相关组件，均为本方案的实现，经过编译打包并导入Docker的密封环境中。

**Bitcomet中间层：**在Android 11中，Anbox的后端及相关模块通过桌面下的Anbox前端产生的QEMU pipe进行通信。

**Bitcomet上层：**Android在容器内缺少图形输出、渲染、音频和键盘设备操作的输入输出功能，其相关功能依赖Anbox的Gralloc、HWC、Audio等相关模块实现。这些模块在QEMU pipe通信的基础上，搭建起对外Linux沟通桥梁，为外部Linux对接内部Android的输入输出、图形输出渲染等功能。外部Anbox针对相关功能实现调用Linux系统库，实现了输入输出渲染等功能。

下面是针对Bitcomet各个部分的详细设计：

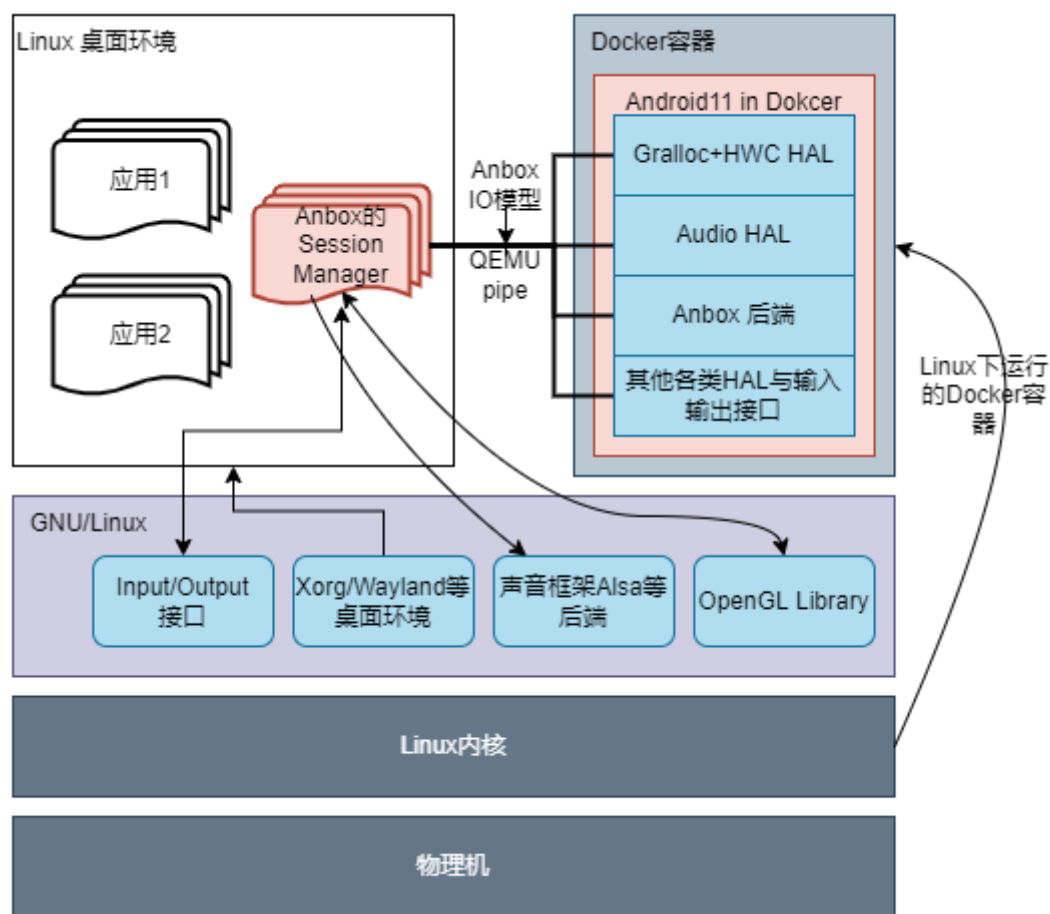


图2.1 Bitcomet方案详细架构图

其中各个部分有相关的实现文档，具体见当前项目下Docs目录内其他文档。

## 2.1 Docker下的安卓11容器设计

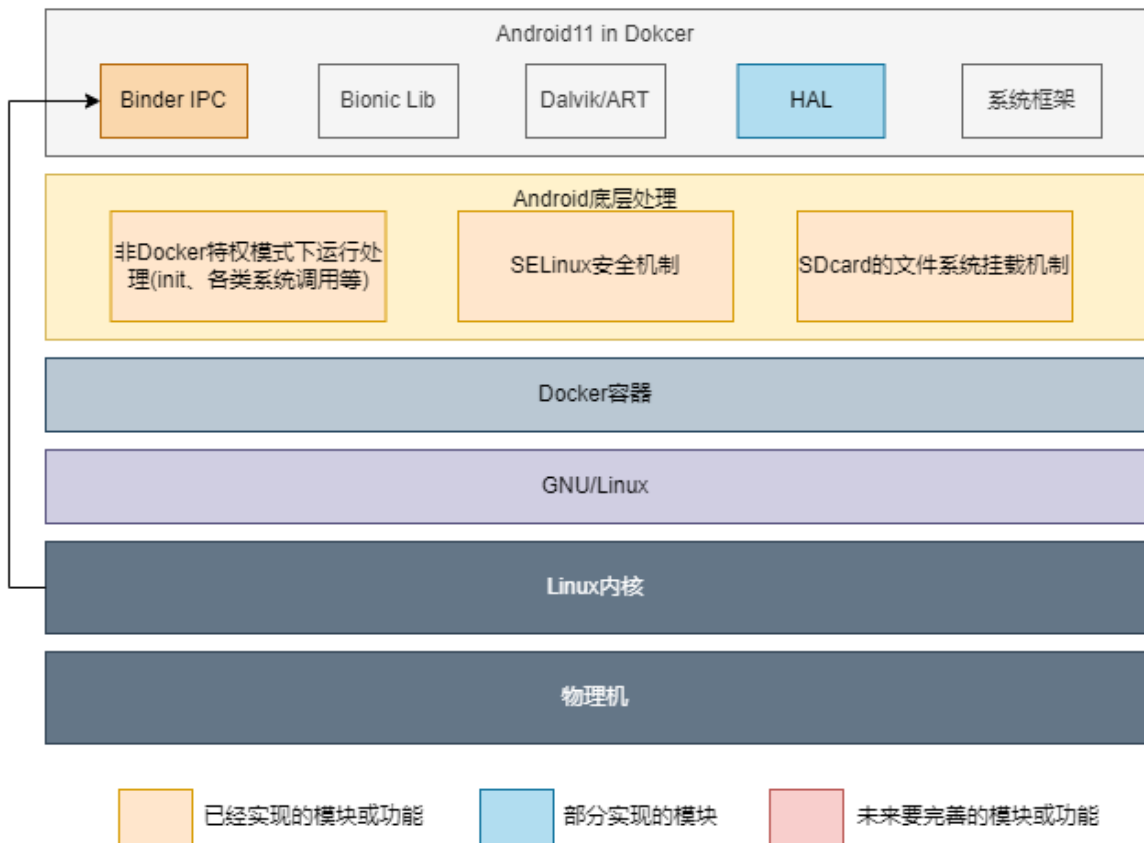


图2.2 Bitcomet实现底层部分设计框架图

### 2.1.1 主要工作

在Bitcomet方案底层部分中，使用Docker容器提供Android运行的环境。当然仅仅提供一个环境还不行，Android 11需要依赖Binder、SELinux、SDcardFS、Ashmem等环境的支持。具体的工作如下：

(1). **内核定制**：Binder与Ashmem需要针对内核定制，目前较新的Linux 5.x已经自带BinderFS和Ashmem模块的支持，旧的内核需要使用Anbox提供的Binder或Ashmem模块。

(2). **SELinux安全机制处理**：本方案选择了对其相关方法进行屏蔽处理，即使实际测试及Docker文档中都反应了Docker下是可以实现这个支持的，但我们仍然选择了屏蔽，其原因如下：

① Docker外是针对指定配置文件的SELinux安全配置，而Android内的还需要单独配置并需要一部分特权。

② 不同平台的Linux内核不一定提供并使用了SELinux安全模块，遇上没有此模块的平台均无法运行。

③ 使用SELinux后，Docker官方提到可能会对性能有一定的损失。

(3). **文件系统挂载处理**：SDcardFS部分是安卓8以后提供的模块，可以让Android接管对SDcardFS下的目录文件的访问权限控制。我们选择了使用FUSE来实现挂载SDcard的虚拟文件系统，FUSE是较旧版本的Android采用的一种方式，其实现主要在用户空间，能提供几乎与SDcardFS相同的功能。但FUSE挂载SDcard的实现也有缺点，其速度比SDcardFS的实现慢，我们仍然选择FUSE方案的原因如下：

① SDcardFS的实现依赖Linux内核实现，不同Linux内核对其支持不一样。

② 由于SDcardFS在内核实现，挂载它也需要一定的特权。



## 2.1.2 Bitcomet底层部分特点

Docker下的安卓11容器设计：

- **高可移植设计**：使用了Docker便集成了Docker相关的优点，Docker相比原有LXC有可移植性、版本控制、回滚、快速部署等优点。同时原有LXC容器与Anbox进行了高度集成，在目前多平台和可移植性考量下，原有Anbox的系统镜像与各个配置选项都不能方便地进行更改，这将会导致我们适配到国产系统中遇到重重困难。
- **快速迭代更新**：使用Docker提供底层容器运行环境，利用其快速部署的优点，可以使Android 11更加灵活地运行在目标平台。
- **优雅的数据管理**：利用Docker的版本控制和回滚特性，设计恢复出厂设置并清空用户数据的功能，并为以后Android镜像更新时能够快速部署，及时把新版本推送。
- **较高系统安全性**：把安卓放在密封的容器当中，其外部访问数据都通过QEMU pipe交给上层Anbox前端应用接管，其应用也只有基本的输入输出功能，安全性较高。
- **极低的运行损耗**：通过容器直接基于现有系统Linux内核运行，并基于Linux内核特性进行容器隔离，这一方案提高了稳定性并降低了运行损耗。

## 2.2 安卓与上层Anbox通信设计

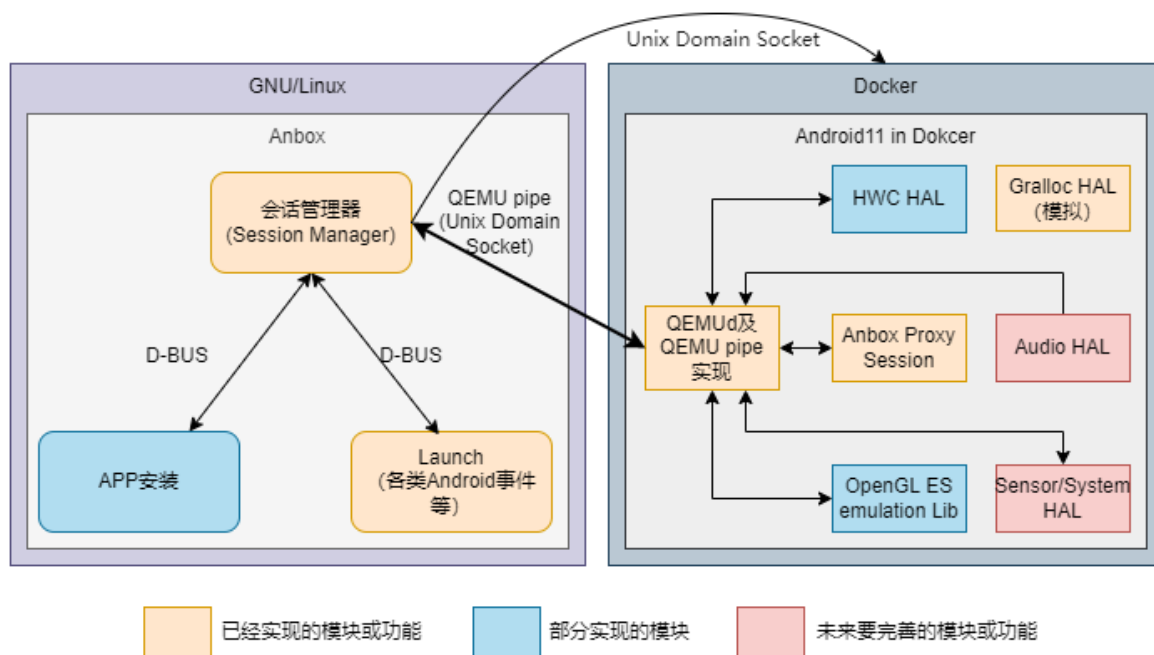


图2.3 Bitcomet中间层部分设计原理图

### 2.2.1 主要工作

在Bitcomet中间层中，我们复用了Anbox这一实现，把其相关实现移植到Android11，并实现其上面的各种功能。我们的主要工作如下：

(1). **通信实现**：把Anbox基于QEMU pipe和qemud的实现移植到Android11，这两个实现的本质均为通过QEMU pipe实现，但由于容器内没有实质上的QEMU pipe设备，因此Anbox魔改了一下使用Unix Domain Socket实现，这种实现从原理上说与QEMU pipe的区别在于：① QEMU pipe设备是通过QEMU的相关模拟实现的，其模拟的设备支持DMA，理论上比Unix Domain Socket这一软件实现更有效率；② Anbox这里相关实现的名字还是叫做QEMU pipe，而QEMU pipe本身也有tcp socket相关的实现。

(2). **设备映射**：在Docker中映射外部Anbox的Session Manager创建的Socket监听到容器内的/dev对应设备，以便容器内的QEMUD或者QEMU pipe相关实现进行调用。

(3). **Anbox通信模块移植**：容器内Anbox为这些实现封装成了叫做HOST\_CONNECTION的方法，容器外Anbox基于谷歌的ProtoBuf实现一种通信模型，其中包括OpenGL ES、HWC、模拟的Surface、Anbox Proxy等组件的RPC调用，以便内部Anbox通过这些封装的方法直接调用，我们也针对这些方法进行了移植和部分测试。

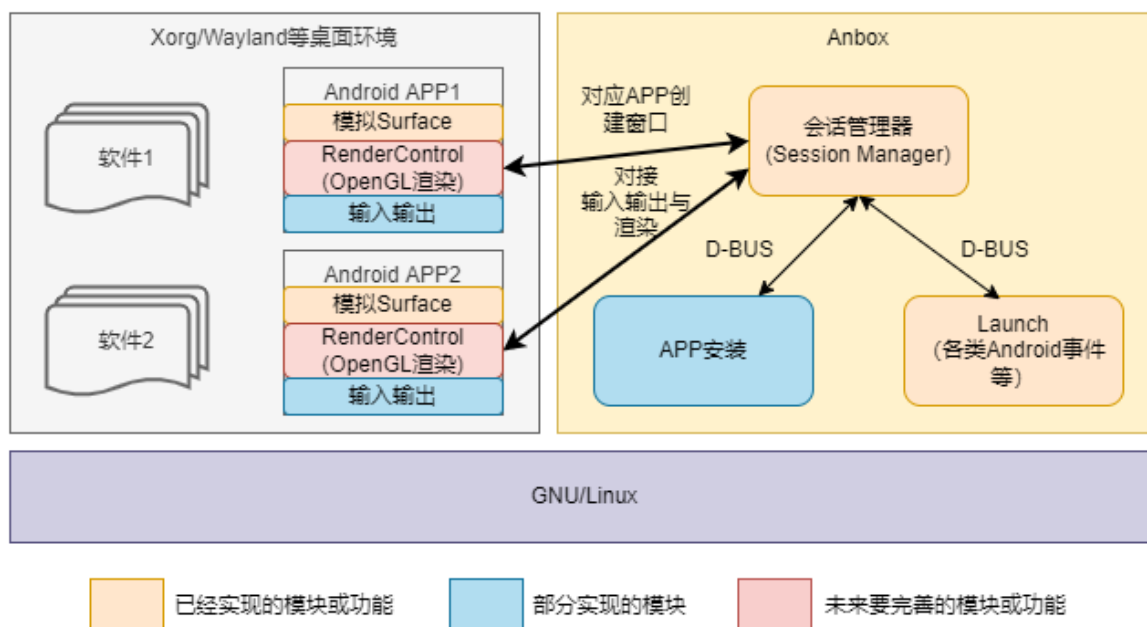
## 2.2.2 Bitcomet中间层部分特点

安卓中运行qemud提供QEMU pipe这一高速管道与上层通信：

- **成熟的通信方案**：这一通信方案基于成熟的Anbox的通信这一部分的功能实现，其上层Session Manager提供接口，由安卓和Anbox两端利用相应API把接口打开进行通信。其实现方案最早可以追溯到2013年的谷歌Goldfish的Android模拟器实现并且沿用至今，实现了Android与Linux的之间通信的兼容性与健壮性并存的实现方式。

- **便捷的交互方式**：本方案在安卓与Linux之间的通信方案设计基于QEMU pipe这一高速通道，直接在Android与Linux之间打通一个灵活的通信渠道。本方案不用考虑需要的具体通信实现，直接在安卓HAL层服务和上层Anbox前端应用中打开相应通道进行安卓与Linux对接的数据通信。

## 2.3 上层Anbox的OpenGL ES渲染以及输入输出设计



### 2.3.1 主要工作

在Bitcomet的上层中，我们也复用Anbox的实现，把其相关部分移植到Android11和Ubuntu 20.04上。Android中对接外部Linux主要在图形输出、渲染、输入输出部分缺失，这些部分需要靠Anbox在中间通信层的部分实现之上，建立各个部分的连接，打通到外部Linux上。我们的主要工作如下：

- (1). **HAL层模块移植**：这一部分主要工作在Android中HAL层的移植以及测试，我们经过在Android工程中对AIDL与HIDL对应HWC与Gralloc部分的模块进行配置，Gralloc的HAL层复用最新Android 11中对安卓模拟器的AVD实现。我们复用Android 11的这一部分主要原因在于其安卓模拟器的实现与Anbox的实现是一样的，都是软件实现的调用ashmem申请图形缓冲区。

- (2). **图形输出部分移植**：从Android的HAL层中Gralloc与HWC模块开始，图形输出部分的核心模块是HWC模块，它负责被SurfaceFlinger调用进行图层合成输出到显示设备，而这里我们选择在Anbox的实现上进行移植修复。该实现主要测试点在于图形HWC，HWC中的发送图层部分开始，通过Anbox的RPC调用，发送图层到外部Anbox的Surface上，需要测试相关图层测试数据是否能到达外部Anbox上合

成输出。

(3). **输入输出部分移植**：在输入部分，Anbox的Session Manager注册了多个Event设备以及audio传输设备，我们需要把这些设备在Docher中映射到/dev下，以便在Android中调用。其中音频部分也复用Android11针对模拟器部分的audio HAL实现，原因在于Anbox这些部分的实现也是复用Android模拟器的实现，其原理是把音频PCM数据通过Anbox的RPC调用方法发送到外部Anbox进行解码播放。

(4). **OpenGL ES部分移植**：Anbox的OpenGL库主要在Android的 /system/lib64/egl 目录下安装了三个动态库，分别为libEGL\_emulation.so、libGLESv1\_CM\_emulation.so和libGLESv2\_emulation.so。Anbox在其中提供虚拟的硬件厂商配置，为Android提供OpenGL ES渲染库。但Anbox实际渲染工作并不是上述三个库文件，这三个库文件作用为采集Android中APP渲染的OpenGL ES指令，并通过高速传输通道qemu-pipe传输将指令传至宿主机Anbox进程中，实际渲染工作由宿主机执行。这部分主要也是移植好Anbox这几个库，检验该三个库与上层Anbox的通信，这部分库也是复用谷歌Android的安卓模拟器AVD实现，主要测试工作在于做相关指令的通信以及实际渲染测试。

### 2.3.2 Bitcomet上层部分特点

Anbox通过QEMU pipe与底层通信，接收安卓的渲染信息渲染以及音频在上层APP输出，把上层前端APP的输入和状态传递给安卓：

- **兼容性高**：主要体现在Linux下对Anbox的兼容性，这一方案的设计把安卓需要与Linux交互和获取的数据都通过QEMU pipe传输给Linux处理，上层Anbox前端不依赖硬件。尤其是渲染也是调用系统现有OpenGL库实现，兼容性相比虚拟机来说更高。
- **易用性高**：通过上层Anbox应用接管底层安卓的输入输出等部分，相当于把安卓应用直接映射给了Linux应用。这一操作安卓应用非常接近使用Linux应用的方式，极大的提高了其易用性，操作起来也更加简单。

## 3. Bitcomet方案测试

由于Bitcomet是一款基于Anbox方案进行移植，实现Android 11系统在Linux平台上运行的产品，其相关资料缺乏，因此市面上并没有针对此类产品进行测试的工具。我们设计了针对该方案设计了基础功能测试与性能测试。具体测试测试过程与结果见“系统基础功能测试与性能分析”文档。

## 4. 未来展望

**未来更加值得选择并研究的方案：**

这里讲述一种未来更有潜力的方案，该方案仍然是使用容器来提供Android运行的环境，这样继承了容器方案的优势，但是在3D渲染和图形输出的部分和输入输出的部分选择了截然不同的方式，把Mesa3D驱动直接移植到Android内，同时在HWC和Gralloc模块上进行处理，直接对接外部Linux实现控制和图形输入输出。此种方法有目前众多新的项目使用，甚至是其Anbox的衍生项目Waydroid以及Redroid、KMRE等方案也是使用了该种方法。直接使用Mesa3D驱动GPU硬件，图形渲染和输入输出能够直接对接GPU硬件，显然可见这种实现方法**性能损失是最低的**。

但是该种方案仍然有**缺点**：Mesa3D驱动即使支持了众多x86和ARM平台下的GPU，包括Intel、AMD、NVIDIA、Qualcomm、ARM等，但仍然有不支持的GPU；Mesa3D是Linux下的库，需要经过许多移植工作才能在Android下工作；针对不同GPU的平台，如果实现Mesa3D的方案，但GPU又未被Mesa3D支持，则相关驱动也需要移植；Mesa3D在Android中部分仅仅只是图形库，在外部Linux内核也需要相应GPU驱动，从而使外部Linux也需要使用Mesa3D库来调用GPU。

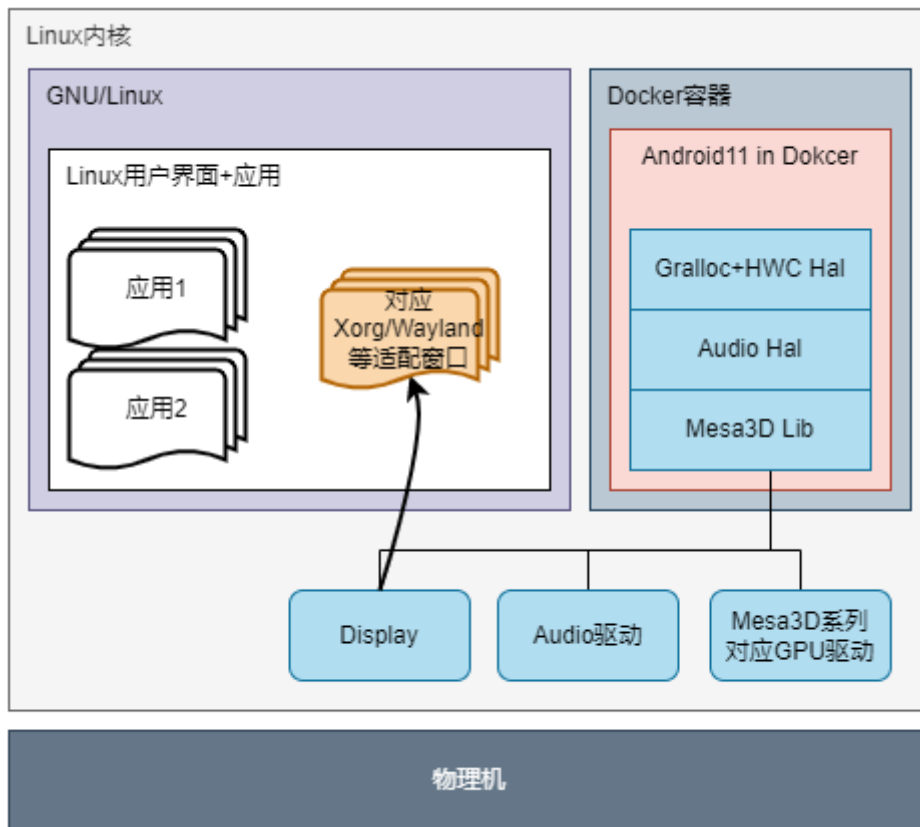


图1.5 Docker运行Android+Mesa3D架构图

本项目Bitcomet是这类移植方式中其中一种兼容性较好的方案，技术仍然在不断发展，目前已有各类新型的，性能损耗更低的技术出现。例如Anbox的衍生项目Waydroid以及KMRE，直接利用GPU库调用底层GPU硬件的方法，实现真机的调用方法，损耗极低。未来本项目发展必然是朝着这些新技术进行研究，并实现对这些新技术的掌握及发展。例如Waydroid方案的支持有局限性，只对接了Wayland实现窗口创建管理及图形输出。而KMRE等是属于商业项目，但从其论文来看，与Waydroid都是同一种实现技术与方案，唯一不同是实现了在Xorg下的输出显示。因此本项目未来需要追赶新技术，实现Wayland、KMRE这类新技术，并且在其之上完善更多图形环境下的适配以及其他各类优化。

同时在这种移植技术的发展上，本项目希望探索与总结这类跨系统平台适配的各类技术的实现与优点，能总结出这种成熟的技术并制定方案，建立成熟的技术体系，提供给更多同类型项目进行参考，助力信息产业国产化，建设更好的国产生态。

## 5. 参考文献

1. 面向桌面Linux的Android运行环境构建. 张超，国防科技大学，2012年10月。
2. Android 架构，Android Open Source Project (google.cn) 链接：<https://source.android.google.cn/devices/architecture?hl=zh-cn>