移动安全

Mobile Security

第三章　移动系统与平台

王寅

天津大学　网络安全学院

wyin@tju.edu.cn

本章目录

AOSP：安卓发展的起源

AOSP 是什么

AOSP（Android Open Source Project）是 Android 开源项目，由 Google 主导维护，任何人都可获取源码，并按许可证修改、编译和使用。

出现契机

Google 希望为厂商、运营商和开发者提供开放的移动平台，减少单一企业对创新的限制。2007 年 Android 与开放手机联盟亮相，2008 年平台源码公开。

主要内容

提供从内核相关代码、HAL 接口、运行时与系统库，到应用框架、系统服务和基础应用的代码，以及构建工具、测试与兼容性资料。

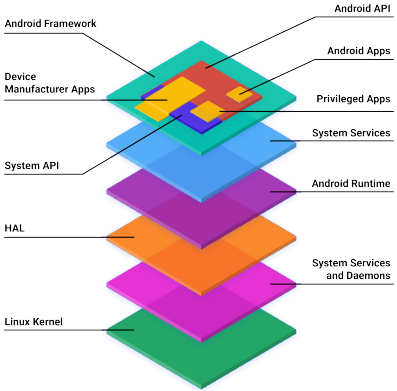
Android 的重要版本

AOSP 的实际应用：厂商系统

厂商以 AOSP 为基础，适配硬件，并修改系统界面、服务和运行策略，加入自有应用与设备互联功能。

- **三星 One UI**：定制 Galaxy 界面与多任务体验，集成 Knox 安全能力。
- **小米 HyperOS**：优化系统运行与界面，加入手机、平板等设备的互联服务。
- **OPPO ColorOS**：扩展分屏与多任务，增加剪贴板隐私保护和跨屏互联。
- **vivo OriginOS**：定制桌面与交互，调整资源调度以保障前台应用流畅度。
- **荣耀 MagicOS**：定制系统交互，加入基于账号的可信互联和安全服务。

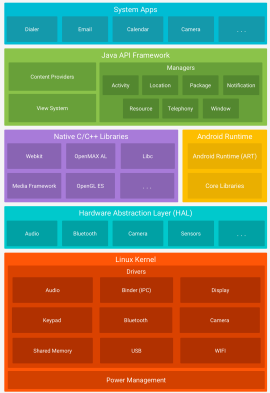
Android 的基本架构



Android 底层内核空间以 Linux Kernel 作为基石，上层用户空间由 Native 系统库、虚拟机运行环境、框架层组成，通过系统调用 (Syscall) 连通系统的内核空间与用户空间。

对于用户空间主要采用 C++ 和 Java 代码编写，通过 JNI 技术打通用户空间的 Java 层和 Native 层 (C++/C)，从而连通整个系统。

Android 的基本架构

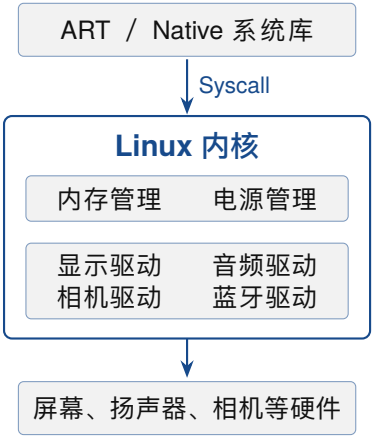


- **App**：面向用户的功能入口。
- **Framework**：组件、界面与系统能力 API。
- **ART / 系统库**：执行代码与提供本地能力。
- **HAL**：约定硬件服务接口。
- **Linux**：调度、内存、驱动与隔离。

A1 Linux 内核层

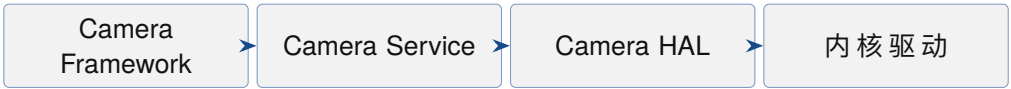
Android 平台的基础是 Linux 内核，比如 ART 虚拟机最终调用底层 Linux 内核来执行功能。Linux 内核的安全机制为 Android 提供相应的保障，也允许设备制造商为内核开发硬件驱动程序。

Linux 内核包括了各种硬件驱动程序，如显示驱动、音频驱动、相机驱动、蓝牙驱动、内存管理和电源管理等。



A2 HAL 硬件抽象层

由于不同移动终端配备的硬件不同，各自使用方式存在巨大差异，因此安卓系统设计了 HAL（Hardware Abstraction Layer），用于约定系统调用硬件的能力。



HAL 封装了标准的接口，供上层的安卓框架使用硬件功能。

例如，HAL 定义了应用框架访问相机硬件的接口，不同的设备制造商可以实现这些接口来控制相机硬件，意味着开发者编写一次代码，即可在多种设备上使用相机功能，而不必考虑底层硬件的差异。

A3 系统运行库层

系统运行库层包括安卓运行时（Android Runtime）环境和一系列的原生 C/C++ 库，包括：

- 虚拟机：Dalvik 虚拟机（Android5 后被 ART 代替）支持已转换为.dex（即“Dalvik Executable”）格式的 Java 应用程序的运行。
- 本地 C/C++ 库：如 WebKit，OpenGL，FreeType 等常用的图形绘制、数据渲染功能。

A4 Framework 框架层

Java API Framework 是创建 Android 应用所需的构建块的基础，可简化核心、模块系统组件和服务的重复使用，包括以下组件和服务：

- **视图系统**，可用于构建应用界面，包括列表、网格、文本框、按钮，甚至可嵌入的网络浏览器。
- **资源管理器**，用于访问非代码资源，例如本地化的字符串、图形和布局文件。
- **通知管理器**，可让所有应用在状态栏中显示自定义提醒。
- **Activity 管理器**，用于管理应用的生命周期，并提供常见的**导航返回堆栈**。
- **Content Provider**，可让应用访问其他应用（例如“通讯录”应用）中的数据或共享自己的数据。

A5 应用层

Android 随附一套用于电子邮件、短信、日历、互联网浏览和通讯录等的核心应用。除此之外，还包括第三方开发的应用，如微信、Facebook 等。

例：拍照请求的完整路径

IPC：进程间的协作

线程是 CPU 调度的最小单元，同时线程是一种有限的系统资源。而**进程一般指一个执行单元**，在 PC 和移动设备上指一个程序或者一个应用。**一个进程可以包含多个线程**，因此进程和线程是包含与被包含的关系。

很多时候，一个进程中需要执行大量耗时的任务，如果这些任务放在主线程中去执行就会造成界面无法响应，严重影响用户体验，在 Android 中有一个特殊的名字叫做 ANR（Application Not Responding），即应用无响应。解决这个问题就需要用到线程，把一些耗时的任务放在线程中完成。

IPC 是 Inter-Process Communication 的缩写，含义为进程间通信或者跨进程通信，是指两个进程之间进行数据交换的过程。

IPC：进程间的协作

例：音乐 App 的界面与播放服务运行在不同进程。

- 用户点击播放，界面进程发送歌曲编号 7 与播放请求。
- 播放进程接收请求，读取歌曲并更新本地播放状态。
- 播放进程返回状态，界面进程更新按钮与进度显示。

问题：进程和线程的区别与联系

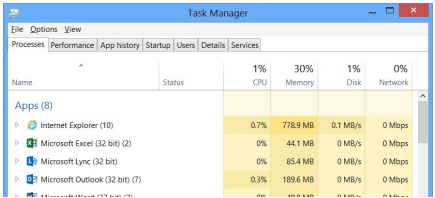
思考

进程和线程的区别与联系是什么？

解析：进程与线程的概念

进程

进程是一个在内存中运行的应用程序。每个进程都有自己独立的一块内存空间，一个进程可以有多个线程。



Windows 任务管理器：进程列表（局部）

线程

线程是进程中的一个执行任务（控制单元），负责当前进程中程序的执行。一个进程至少有一个线程，一个进程可以运行多个线程，多个线程可共享数据。

例：一个相册进程中

主线程响应滑动并更新界面；工作线程读取、解码图片。两个线程共享图片数据，各自保存执行位置与调用栈。

以 JVM 为例：同一进程中的多个线程共享堆和方法区资源，每个线程有自己的程序计数器、虚拟机栈和本地方法栈。

解析：进程和线程的区别与联系

线程具有传统进程的执行特征，常称轻型进程（LWP）；传统单线程进程可称重型进程（HWP）。现代运行中的进程至少包含一个线程，也可包含多个线程。

根本区别	进程是资源分配的基本单位；线程是任务调度和执行的基本单位。
资源开销	进程有独立的程序上下文，创建与切换开销通常较大；同一进程的线程共享代码和数据，各有运行栈与程序计数器，开销通常较小。
包含关系	线程是进程的一部分。一个进程可有多个执行序列，由多个线程协作完成任务。
内存分配	同一进程的线程共享地址空间和资源；不同进程的地址空间相互独立，也可显式建立共享内存。
影响关系	进程崩溃通常不会直接破坏其他进程的私有内存；线程发生致命错误可能终止整个进程。多进程提供更强的故障隔离。
执行过程	程序启动时建立进程及初始线程，线程各有执行入口与序列，并依托进程资源运行。不同进程中的线程和同进程的线程均可并发执行。

B1.1 序列化的概念

序列化将对象中需要传递的状态编码为可传输或保存的数据；**反序列化**按约定读取数据，重建对象。

进程拥有各自的地址空间，另一个进程无法直接使用本进程的普通对象引用。双方需要约定字段、类型和读取方式。

传递阶段	例：传递一条联系人记录
发送方	对象中包含 id=7、name=“小王”
编码与传输	将编号和姓名按约定写入数据，交给 IPC 机制传递
接收方	读取字段，创建本地联系人对象，用于显示或查询

序列化负责表达数据，Binder 或 Socket 等机制负责传输。

B1.2 Serializable 与 Parcelable

Serializable：交给对象流处理

```
class SUser implements Serializable {
    int id;
    ...
}
// 对象流自动处理字段
out.writeObject(user); // 写入对象
SUser copy = (SUser) in.readObject();
```

只做标记：接口本身没有必须实现的方法。默认由对象流遍历可序列化字段，完成保存与恢复。

像“自动打包”：开发者通常不用逐个写字段，代码较少；对象流需要处理类型和对象关系，通常开销较大。

Parcelable：明确字段如何读写

```
class PUser implements Parcelable {
    int id;
    public void writeToParcel(Parcel p,
    ...) {
        p.writeInt(id);  // 写字段
    }
    PUser(Parcel p) { id = p.readInt(); }
    ... // CREATOR 等其余实现
}
```

明确读写规则：逐项写入字段，按相同类型和顺序读取，由 CREATOR 重建对象。

像“按清单装箱”：适合 Android 组件间传递；读写规则需自行维护，也可由工具生成。

B1.3 序列化的安全风险

```
// readObject：恢复对象时调用的自定义钩子
private void readObject(ObjectInputStream in)
    throws IOException, ClassNotFoundException {
  in.defaultReadObject();  // 从对象流恢复字段
  new File(path).delete(); // 删除path指向的文件
}
```

执行非预期逻辑

path 来自外部对象流，恢复对象时立即删除该路径的文件。

```
int count = parcel.readInt(); // 读取长度
byte[] payload = new byte[count]; // 按长度分配
parcel.readByteArray(payload); // 读入字节数组
```

崩溃与资源耗尽

count 未经校验。负值会抛异常，极大值可能耗尽内存。

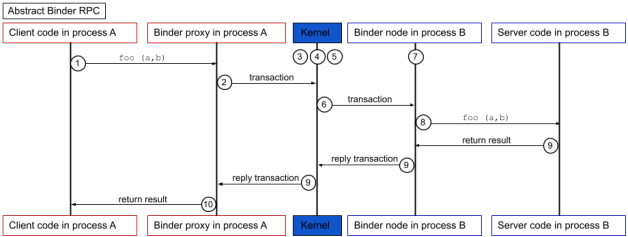
```
class Session implements Serializable {
  String accessToken = "demo-token";
}
// writeObject：将对象字段写入流，不会自动加密
out.writeObject(new Session());
```

敏感数据泄露

令牌随字段写入对象流，未自动加密，输出泄露会暴露令牌。

B2.1 Binder 的概念

Binder 是 Android 的一套进程间通信机制，由应用侧接口、本地通信支持与内核驱动等部分协同完成请求传递。



客户端把参数写入 Parcel 并发出请求；Binder 传递事务，服务端处理后将结果写入回复。

B2.2 简单 Binder 服务：计算两数之和（服务端）

```
@Override
protected boolean onTransact(int code, Parcel in,
    Parcel out, int flags) throws RemoteException
{
  if (code != ADD)
    return super.onTransact(
        code, in, out, flags);
  in.enforceInterface(TOKEN);
  int a = in.readInt(), b = in.readInt();
  out.writeNoException();
  out.writeInt(a + b);
  return true;
}
```

服务端处理请求

- ADD 表示“加法”操作。
- 按约定读取两个整数。
- 将计算结果写入回复。

AddService 的 onBind 返回实现该方法的 Binder 对象。

例：读入 2 和 3，回复 5。

B2.2 简单 Binder 服务：计算两数之和（客户端）

```
Parcel data = Parcel.obtain();
Parcel reply = Parcel.obtain();
try {
  data.writeInterfaceToken(TOKEN);
  data.writeInt(2); data.writeInt(3);
  boolean ok = remote.transact(
      ADD, data, reply, 0);
  if (!ok)
    throw new RemoteException("Unknown request");
  reply.readException();
  return reply.readInt(); // 5
} finally {
  data.recycle(); reply.recycle();
}
```

客户端发起请求

- remote 来自绑定成功后的回调。
- 先写接口标识，再写 2 和 3。
- transact 发送并等待回复。
- 读取结果，最后回收 Parcel。

结果：add(remote) 返回 5。

B3.1 Socket 的概念

Socket 是程序进行通信时使用的端点接口，可用来建立本地或网络数据通道。

类型	通信方式	举例
TCP Socket	有连接、有序的字节流	聊天程序先发送消息长度，再发送对应字节的聊天文本
UDP Socket	以数据报为单位发送	游戏发送角色位置并附序号，接收方丢弃迟到的旧位置
LocalSocket	同一设备上的本地通道	本机工具连接本地服务，发送“查询状态”并接收运行结果

TCP 例子：本机两个进程计算 2+3

1. 服务端监听 127.0.0.1:5050，调用 accept 等待连接；客户端用 connect 连接该地址。
2. 客户端依次发送 32 位整数 2 和 3，共 8 字节；服务端按相同顺序读取并相加。
3. 服务端回复 32 位整数 5，共 4 字节；客户端读取结果后，双方关闭本次连接。

B3.2 简单 Socket 服务：计算两数之和

服务端：接收并回复

```
try (Socket s = listener.accept()) {
  s.setSoTimeout(2000);
  DataInputStream in =
      new DataInputStream(
          s.getInputStream());
  DataOutputStream out =
      new DataOutputStream(
          s.getOutputStream());
  int a = in.readInt();
  int b = in.readInt();
  out.writeInt(a + b);
  out.flush();
}
```

客户端：连接并请求

```
try (Socket s = new Socket()) {
  s.connect(new InetSocketAddress(
      "127.0.0.1", port), 2000);
  s.setSoTimeout(2000);
  DataOutputStream out =
      new DataOutputStream(
          s.getOutputStream());
  out.writeInt(2); out.writeInt(3);
  out.flush();
  DataInputStream in =
      new DataInputStream(
          s.getInputStream());
  return in.readInt(); // 5
}
```

协议：发送两个 32 位整数，回复一个整数；示例在本机 5050 端口通信。

Binder 和 Socket 的区别

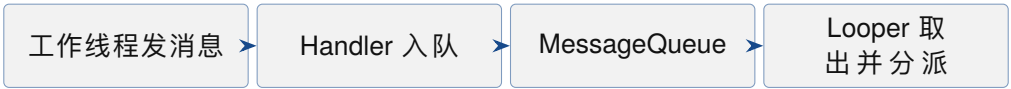
Binder 侧重同一设备上的服务调用；Socket 提供本地或跨设备的数据通道。

比较维度	Binder	Socket
通信范围	同一 Android 设备上的跨进程调用	TCP/UDP 可跨设备；LocalSocket 用于本机
调用方式	以事务传递操作编号和 Parcel 参数，服务端分派处理并返回结果	通过字节流或数据报传数据，双方约定消息格式及请求、回复规则
身份校验	内核提供调用方 UID，服务端据此检查权限	网络连接需应用层认证；本地 Socket 可获取对端进程凭据
故障感知	通过远端进程死亡通知感知	通过流结束、错误或超时感知
传输开销	针对本机短请求优化，驱动将事务数据复制到接收进程的映射缓冲区	经 Socket 缓冲区传输；开销取决于协议、数据量与实现

B4.1 Handler 的概念

Handler 是与一个 Looper 关联的消息处理接口，通过 post 或 sendMessage 向目标线程的消息队列提交任务。

Looper 负责运行线程的消息循环：从队列取出可执行的消息，交给对应的 Handler 处理；没有可执行消息时等待。



- MessageQueue 保存待处理消息；Handler 的回调在所绑定 Looper 的线程执行。
- 主线程的 Looper 由系统准备，负责分派界面相关任务；耗时工作应交给工作线程。

B4.2 简单 Handler 示例：查询后更新界面

```
// ui 绑定主线程的 Looper
Handler ui = new Handler(
    Looper.getMainLooper());

executor.execute(() -> {
    // 工作线程：查询天气
    String result = queryWeather();
    ui.post(() -> {
        // 主线程：更新界面
        label.setText(result);
    });
});
```

- **指定主线程**：getMainLooper() 让 ui 关联主线程，交给它的任务将在主线程执行。
- **后台查天气**：executor 把 queryWeather() 放到工作线程，查询结果存入 result。
- **提交界面更新**：ui.post 将“修改文字”的任务加入主线程队列，等待执行。
- **显示查询结果**：主线程执行 label.setText(result)，把天气文字显示在 TextView 中。

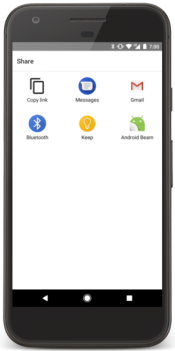
queryWeather() 在 executor 的工作线程执行；label.setText() 在主线程执行。

C1 PMS：应用包与组件管理

PMS（PackageManagerService）维护已安装应用及其组件的信息。

- **包的管理**：协调安装、更新与卸载，记录包名、版本、签名及状态。
- **组件查询**：根据 Manifest 与 Intent 条件，查找能处理请求的组件。

例：分享文字时，系统查询可接收该内容的应用，让用户选择目标。



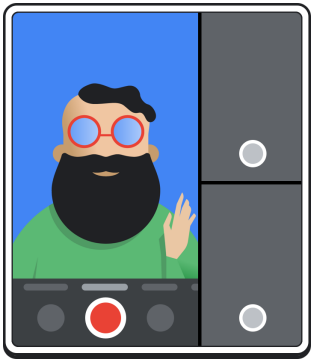
应用选择界面

C2 WMS：窗口管理

WMS（WindowManagerService）管理窗口布局、可见性、层级与焦点。

- **布局与显示**：管理窗口的位置、大小及可见状态，适应旋转与分屏。
- **层级与焦点**：安排窗口的前后关系，维护当前获得焦点的窗口。

例：进入分屏模式时，系统调整窗口大小和位置，让多个应用同时显示。



同一屏幕上的多个窗口

C3 IMS：输入事件管理

IMS（InputManagerService）管理输入设备，与底层组件协作处理触摸、键盘和鼠标事件。

- **设备管理**：跟踪输入设备的连接与配置，让系统识别可用输入方式。
- **事件分派**：结合窗口位置、焦点与策略，将事件交给相应窗口。

例：点击按钮时，输入系统将触摸事件送到目标窗口，应用再响应按钮操作。

输入事件分派到目标窗口

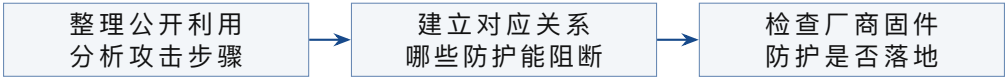
研究：厂商内核装上了哪些防护?

Defects-in-Depth: Analyzing the Integration of Effective Defenses against One-Day Exploits in Android Kernels

发表：USENIX Security 2024
作者：Lukas Maar 等
单位：格拉茨技术大学；A-SIT Austria

解决的问题：上游 Android 内核已有防护，厂商定制系统是否真正启用？当已知漏洞尚未修补时，这些防护还能否阻止攻击者取得高权限?

提出的方法：把公开漏洞利用拆成步骤，找出能中断这些步骤的防护，再检查厂商固件是否包含并有效实现它们。



做出的贡献：分析 26 个公开利用、994 份设备固件。上游防护可阻断 84.6% 的已分析利用流程；各厂商的阻断比例为 28.8%—54.6%。研究量化了上游与厂商实际防护之间的差距。

研究：怎样测试闭源 Binder 服务?

NASS: Fuzzing All Native Android System Services with Interface Awareness and Coverage

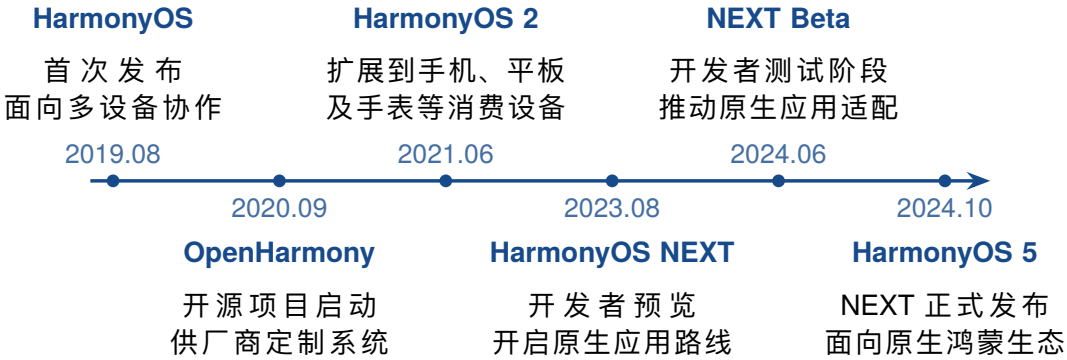
发表：USENIX Security 2025
作者：Philipp Mao、Marcel Busch、Mathias Payer
单位：洛桑联邦理工学院（EPFL）

解决的问题：App 通过 Binder 调用系统服务，服务端必须处理不可信参数。厂商原生服务常不开源，测试者连参数的类型和顺序都不知道，随便填字节很难深入测试。

提出的方法：观察服务端怎样读取 Parcel，推断参数格式；按格式生成并改变请求，优先保留能触达新代码路径的输入。这种反复变换输入的测试称为模糊测试。

做出的贡献：把“恢复接口”和“跟踪请求执行路径”结合，支持无源码测试。作者在 Google、三星、小米和一加设备上发现 12 个独立缺陷，论文发表时已有 5 个 CVE 编号。

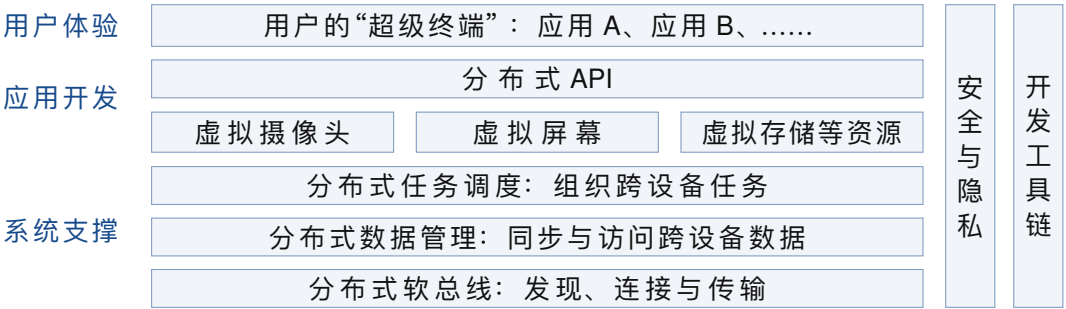
鸿蒙系统的发展



OpenHarmony 由开放原子开源基金会孵化运营，提供可供厂商定制的开源系统。

HarmonyOS 是华为的商业系统，NEXT 阶段推动原生鸿蒙应用生态建设。

鸿蒙的抽象架构

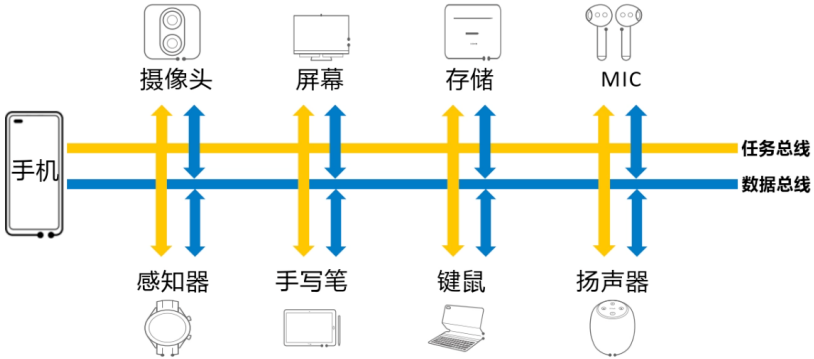


例：视频通话中，手机调度，平板采集画面，大屏显示，音箱播放声音。

OpenHarmony 的系统架构

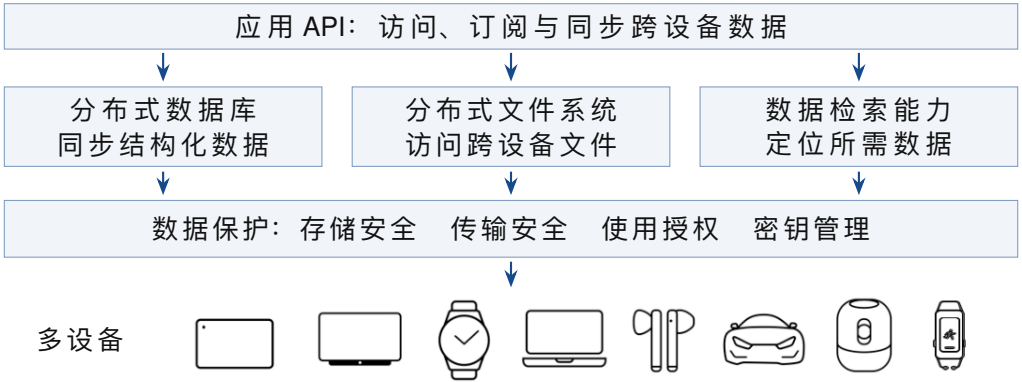
分布式设计：软总线

分布式软总线通过统一的通信协议和设备发现机制，可以实现多设备间的高速、低延迟通信。软总线屏蔽了底层通信协议的差异，为上层应用提供统一的分布式通信接口。



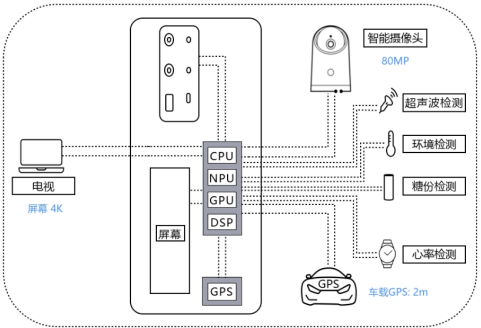
分布式设计：数据管理

分布式数据管理让应用在可信设备间同步和访问数据，保持跨设备使用时的业务连续性。



分布式设计：硬件接入

分布式硬件将其他设备的可用硬件接入本机系统，使应用通过熟悉的接口使用远端能力。



- **资源管理**：发现、记录和查询可信设备提供的硬件。
- **硬件虚拟化**：使能对应部件，在本机形成可调用的虚拟设备。
- **远端执行**：控制请求送到目标设备，采集结果或媒体流返回本机。

ArkTS：应用开发语言

ArkTS 保留 TypeScript 的基本语法风格，强化静态检查，并扩展应用开发能力。

- **约束动态特性**：限制 any、动态增删对象属性等用法，让类型与对象结构更明确。
- **保留常用语法**：使用类、接口、函数和泛型，配合类型与空值检查。
- **扩展开发能力**：支持 ArkUI 声明式语法、状态装饰器，以及并发相关能力。

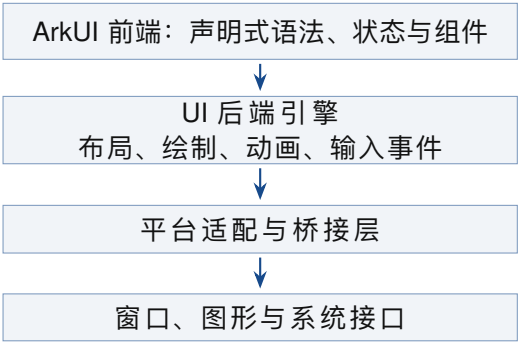
```
class User {
  name: string = "";
}

function greet(user: User): string {
  return `你好，${user.name}`;
}
```

例：name 声明为字符串并初始化，greet 只接受 User 对象，类型问题可在开发期发现。

ArkUI：声明式界面框架

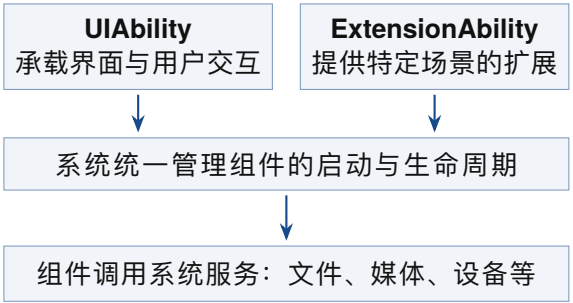
ArkUI 提供界面组件、布局、动画和输入事件处理，声明式界面根据应用状态更新。




```
@Entry
@Component
struct CounterPage {
  @State count: number = 0;
  build() {
    Column() {
      Text(`次数：${this.count}`)
      Button("加一")
        .onClick(() => {
          this.count++;
        })
    }
  }
}
```

Ability：应用功能组件

Ability 按使用场景组织应用功能，组件的启动与生命周期由系统管理。



- **UIAbility**：展示界面并与用户交互，如图库浏览或支付收款。
- **ExtensionAbility**：按特定场景扩展能力，如桌面卡片、输入法。
- **组件协作**：按规则提供调用入口、传递数据，并适配目标设备。

例：相册用 UIAbility 展示照片，用扩展组件提供桌面卡片，随 HAP 打包。

鸿蒙应用包：HAP 是什么

HAP（Harmony Ability Package）是鸿蒙应用安装的基本单位。

- **实现功能**：Ability 类型的 Module 用于实现应用的功能和特性。
- **编译成包**：每个 Ability 类型的 Module 编译后，生成一个以**.hap** 为后缀的文件，称为 **HAP** 包。
- **组成应用**：一个应用可以包含一个或多个 **HAP** 包。

鸿蒙应用包：HAP 的分类

HAP 主要分为两种类型：

- **entry 类型的 Module**：应用的主模块，包含应用的入口界面、入口图标和主功能特性，编译后生成 entry 类型的 **HAP**。每一个应用分发到同一类型的设备上的应用程序包，只能包含唯一一个 entry 类型的 **HAP**，也可以不包含。
- **feature 类型的 Module**：应用的动态特性模块，编译后生成 feature 类型的 **HAP**。一个应用中可以包含一个或多个 feature 类型的 **HAP**，也可以不包含。

鸿蒙应用包：HAR 与 HSP

Library 类型的 Module：用于实现代码和资源的共享。同一个 Library 类型的 Module 可以被其他的 Module 多次引用，合理地使用该类型的 Module，能够降低开发和维护成本。Library 类型的 Module 分为 Static 和 Shared 两种类型，编译后生成共享包。

- **Static Library**：静态共享库。编译后生成一个以.har 为后缀的文件，即静态共享包 **HAR**（Harmony Archive）。
- **Shared Library**：动态共享库。编译后生成一个以.hsp 为后缀的文件，即动态共享包 **HSP**（Harmony Shared Package）。

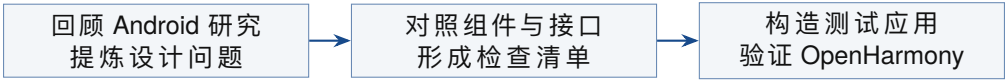
研究：新系统会重现旧的设计问题吗？

SoK: History Doesn't Repeat Itself, but Android Design-Level Vulnerabilities Rhyme in OpenHarmony

发表：USENIX Security 2026
作者：Hongkai Chen 等
单位：亚利桑那州立大学；CISPA；俄亥俄州立大学

解决的问题：Ability 等组件采用新的代码实现后，相似的组件调用与权限设计，是否仍会遇到 Android 已经出现过的问题？研究把历史漏洞经验变成检查清单。

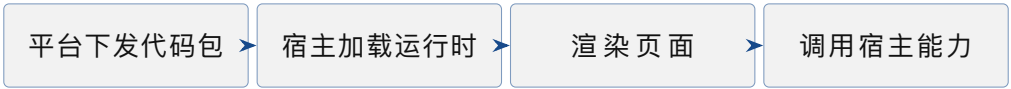
提出的方法：整理 116 篇研究，提炼 56 种设计缺陷及审计方法；把 Android 机制对应到 OpenHarmony 机制，再用测试应用逐项验证。



做出的贡献：在 OpenHarmony 4.1/5.1 中验证出 24 种设计缺陷。例如，发起 Ability 调用时只描述“做什么”、未指定接收组件，可能被声明同类动作的恶意应用接走。

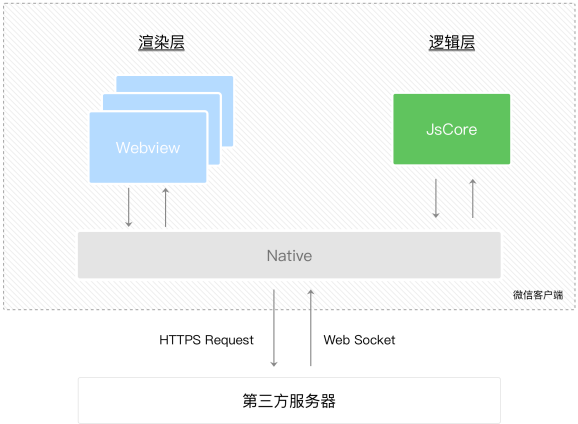
小程序平台

小程序以代码包交付，由微信、支付宝、抖音等宿主提供运行环境。



- 用户通常无需单独安装一个完整 App。
- 开发者复用宿主的登录、支付、分享和设备接口。
- 代码包仍会下载或缓存，运行能力受宿主控制。

微信的逻辑层与渲染层



- 逻辑层执行 JavaScript 业务代码。
- 渲染层显示模板、样式与组件。
- 宿主中转数据、事件和原生能力调用。

小程序的用户信息管理

以微信小程序为例，平台分别提供身份识别、资料填写和手机号获取能力。

用户信息	平台 API / 组件	获取方式与用途
用户标识	`wx.login()` 服务端 `code2Session`	前端取得 code，服务端换取 OpenID，识别本小程序用户；符合条件时返回 UnionID，关联同一开放平台账号下的用户。
头像与昵称	`button: chooseAvatar` `input: type="nickname"`	用户主动选择头像、填写昵称；小程序接收头像临时路径和昵称，保存到自己的用户资料中。
手机号	`button: getPhoneNumber` 服务端手机号获取接口	用户点击并同意后取得动态 code；服务端换取手机号，用于账号绑定、联系等业务。

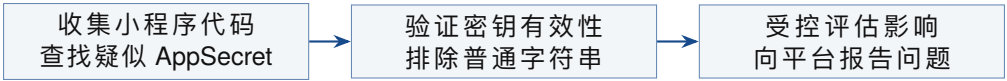
研究：AppSecret 为何必须留在服务端?

Don’t Leak Your Keys: Understanding, Measuring, and Exploiting the AppSecret Leaks in Mini-Programs

发表：ACM CCS 2023
作者：Yue Zhang、Yuqing Yang、Zhiqiang Lin
单位：俄亥俄州立大学（论文署名单位）

解决的问题：AppSecret 是小程序向平台证明身份的“钥匙”。如果开发者把它放进下发给用户的前端代码包，其他人拿到代码，也可能冒用小程序身份访问平台能力。

提出的方法：批量收集代码包，按字符串特征查找疑似密钥，再通过平台接口确认是否有效；在受控账号中评估泄露后能够造成的影响。



做出的贡献：在 3,450,586 个微信小程序中发现 40,880 个泄露 AppSecret。研究揭示账号冒用和付费服务滥用等风险，并推动平台补充数据校验接口。

对比：三类平台的分析入口

维度	Android 原生 App	小程序	鸿蒙原生 App
运行载体	操作系统管理的应用进程	宿主提供的运行时	鸿蒙应用进程
组件与界面	Activity 等＋ View / Compose	Page ＋平台组件	UIAbility ＋ ArkUI
能力调用	框架 API 与系统服务	宿主 API 与平台规则	系统 Kit 与 Ability 接口
首要定位	UID、组件、Binder、线程	小程序身份、宿主、基础库	组件、模块、权限、设备能力

本章小结

Android 系统架构

AOSP
分层系统架构
IPC 通信
PMS/WMS/IMS

鸿蒙系统

HarmonyOS
分布式设计
ArkTS 与 ArkUI
Ability
HAP

HarmonyOS

小程序平台

小程序架构
平台能力与信息管理

微信公众平台 | 小程序