* 基础知识

Android 应用采用 Java 编程语言编写。Android SDK 工具将您的代码 — 连同任何数据和资源文件 — 编译到一个 APK：*Android 软件包*，即带有 .apk 后缀的存档文件中。一个 APK 文件包含 Android 应用的所有内容，它是基于 Android 系统的设备用来安装应用的文件。

安装到设备后，每个 Android 应用都运行在自己的安全沙箱内：

* Android 操作系统是一种多用户 Linux 系统，其中的每个应用都是一个不同的用户；
* 默认情况下，系统会为每个应用分配一个唯一的 Linux 用户 ID（该 ID 仅由系统使用，应用并不知晓）。系统为应用中的所有文件设置权限，使得只有分配给该应用的用户 ID 才能访问这些文件；
* 每个进程都具有自己的虚拟机 (VM)，因此应用代码是在与其他应用隔离的环境中运行；
* 默认情况下，每个应用都在其自己的 Linux 进程内运行。Android 会在需要执行任何应用组件时启动该进程，然后在不再需要该进程或系统必须为其他应用恢复内存时关闭该进程。

Android 系统可以通过这种方式实现*最小权限原则*。也就是说，默认情况下，每个应用都只能访问执行其工作所需的组件，而不能访问其他组件。 这样便营造出一个非常安全的环境，在这个环境中，应用无法访问系统中其未获得权限的部分。

不过，应用仍然可以通过一些途径与其他应用共享数据以及访问系统服务：

* 可以安排两个应用共享同一 Linux 用户 ID，在这种情况下，它们能够相互访问彼此的文件。 为了节省系统资源，可以安排具有相同用户 ID 的应用在同一 Linux 进程中运行，并共享同一 VM（应用还必须使用相同的证书签署）。
* 应用可以请求访问设备数据（如用户的联系人、短信、可装载存储装置 [SD 卡]、相机、蓝牙等）的权限。 用户必须明确授予这些权限。 如需了解详细信息，请参阅 [使用系统权限](https://developer.android.com/training/permissions/index.html)。
* **系统权限**

Android 是一个权限分隔的操作系统，其中每个应用都有其独特的系统标识（Linux 用户 ID 和组 ID）。系统各部分也分隔为不同的标识。Linux 据此将不同的应用以及应用与系统分隔开来。

其他更详细的安全功能通过“权限”机制提供，此机制会限制特定进程可以执行的具体操作，并且根据 URI 权限授权临时访问特定的数据段。

本文档介绍应用开发者可以如何使用 Android 提供的安全功能。一般性的 [Android 安全性概览](http://source.android.com/tech/security/index.html)在“Android 开源项目”中提供。

## 安全架构

Android 安全架构的中心设计点是：在默认情况下任何应用都没有权限执行对其他应用、操作系统或用户有不利影响的任何操作。这包括读取或写入用户的私有数据（例如联系人或电子邮件）、读取或写入其他应用程序的文件、执行网络访问、使设备保持唤醒状态等。

由于每个 Android 应用都是在进程沙盒中运行，因此应用必须显式共享资源和数据。它们的方法是声明需要哪些权限来获取基本沙盒未提供的额外功能。应用以静态方式声明它们需要的权限，然后 Android 系统提示用户同意。

应用沙盒不依赖用于开发应用的技术。特别是，Dalvik VM 不是安全边界，任何应用都可运行原生代码（请参阅 [Android NDK](https://developer.android.com/tools/sdk/ndk/index.html)）。各类应用 — Java、原生和混合 — 以同样的方式放在沙盒中，彼此采用相同程度的安全防护。

## 应用签署

所有 APK（.apk 文件）都必须使用证书签署，其私钥由开发者持有。此证书用于识别应用的作者。证书不需要由证书颁发机构签署；Android 应用在理想情况下可以而且通常也是使用自签名证书。证书在 Android 中的作用是识别应用的作者。这允许系统授予或拒绝应用对[签名级权限](https://developer.android.com/guide/topics/manifest/permission-element.html" \l "plevel)的访问，以及授予或拒绝应用[获得与另一应用相同的 Linux 身份的请求](https://developer.android.com/guide/topics/manifest/manifest-element.html#uid)。

## 用户 ID 和文件访问

在安装时，Android 为每个软件包提供唯一的 Linux 用户 ID。此 ID 在软件包在该设备上的使用寿命期间保持不变。在不同设备上，相同软件包可能有不同的 UID；重要的是每个软件包在指定设备上的 UID 是唯一的。

由于在进程级实施安全性，因此任何两个软件包的代码通常都不能在同一进程中运行，因为它们需要作为不同的 Linux 用户运行。您可以在每个软件包的 AndroidManifest.xml 的 [manifest](https://developer.android.com/reference/android/R.styleable.html#AndroidManifest) 标记中使用 [sharedUserId](https://developer.android.com/reference/android/R.attr.html" \l "sharedUserId) 属性，为它们分配相同的用户 ID。这样做以后，出于安全目的，两个软件包将被视为同一个应用，具有相同的用户 ID 和文件权限。请注意，为保持安全性，只有两个签署了相同签名（并且请求相同的 sharedUserId）的应用才被分配同一用户 ID。

应用存储的任何数据都会被分配该应用的用户 ID，并且其他软件包通常无法访问这些数据。使用 [getSharedPreferences(String, int)](https://developer.android.com/reference/android/content/Context.html" \l "getSharedPreferences(java.lang.String, int))、[openFileOutput(String, int)](https://developer.android.com/reference/android/content/Context.html" \l "openFileOutput(java.lang.String, int)) 或 [openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory)](https://developer.android.com/reference/android/content/Context.html" \l "openOrCreateDatabase(java.lang.String, int, android.database.sqlite.SQLiteDatabase.CursorFactory)) 创建新文件时，可以使用 [MODE\_WORLD\_READABLE](https://developer.android.com/reference/android/content/Context.html#MODE_WORLD_READABLE) 和/或 [MODE\_WORLD\_WRITEABLE](https://developer.android.com/reference/android/content/Context.html#MODE_WORLD_WRITEABLE) 标记允许任何其他软件包读取/写入文件。设置这些标记时，文件仍归您的应用所有，但其全局读取和/或写入权限已适当设置，使任何其他应用都可看见它。

## URI 权限

到目前为止所述的是标准权限系统，内容提供程序仅仅使用此系统通常是不够的。内容提供程序可能需要通过读取和写入权限保护自己，而其直接客户端也需要将特定 URI 传给其他应用以便于它们运行。邮件应用中的附件是一个典型的示例。应通过权限保护对邮件的访问，因为这是敏感的用户数据。但是，如果将图像附件的 URI 提供给图像查看程序，该图像查看程序不会有打开附件的权限，因为它没有理由拥有访问所有电子邮件的权限。

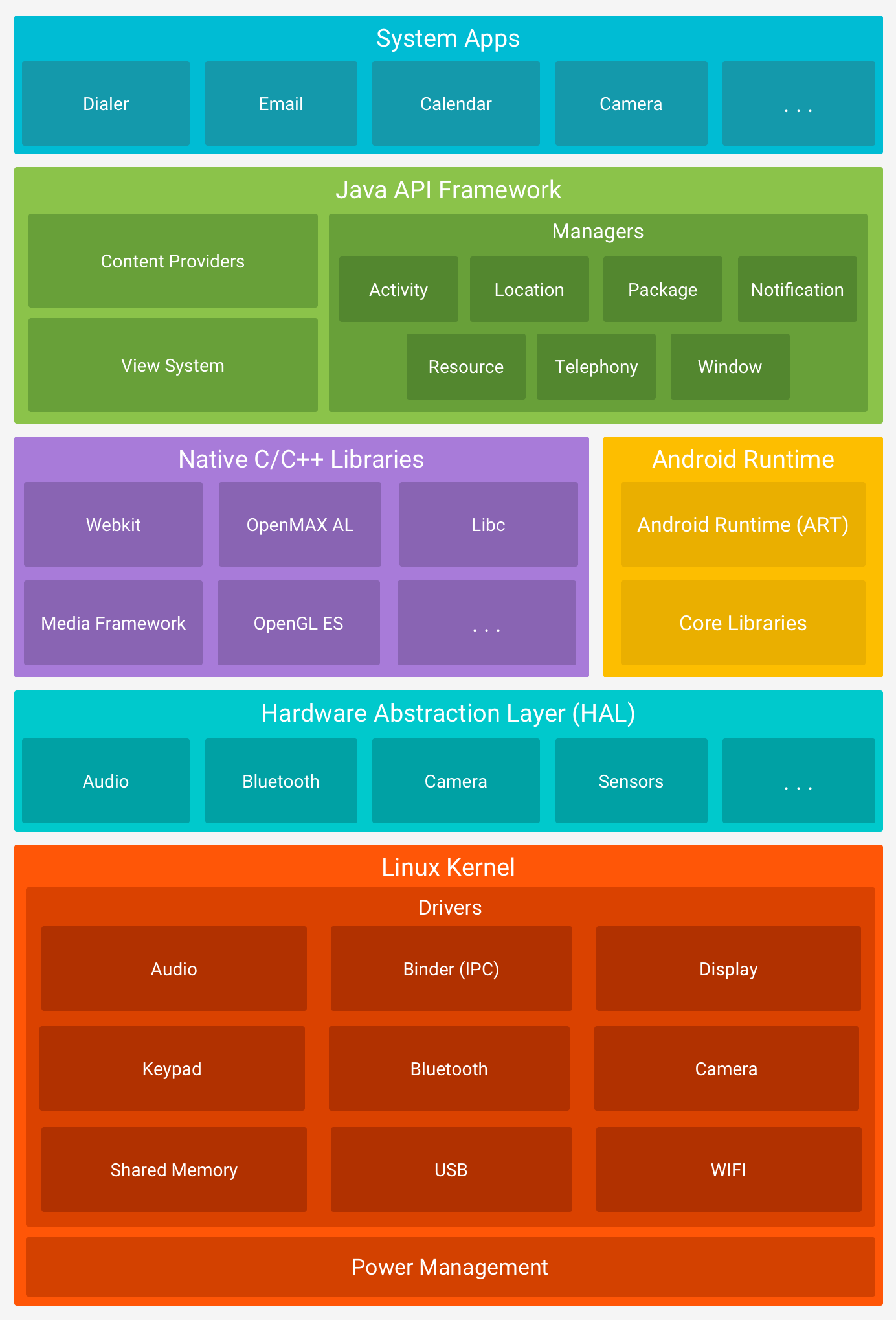
此问题的解决方法是采用 per-URI 权限机制：在启动 Activity 或返回结果给 Activity 时，调用方可以设置 [Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION](https://developer.android.com/reference/android/content/Intent.html" \l "FLAG_GRANT_READ_URI_PERMISSION) 和/或 [Intent.FLAG\_GRANT\_WRITE\_URI\_PERMISSION](https://developer.android.com/reference/android/content/Intent.html" \l "FLAG_GRANT_WRITE_URI_PERMISSION)。这将授予接收 Activity 权限访问 intent 中的特定数据 URI，而不管它是否具有访问 intent 对应的内容提供程序中数据的任何权限。

此机制支持常见的能力式模型，其中用户交互（打开附件、从列表中选择联系人等）驱动临时授予细化的权限。这是一项关键功能，可将应用所需的权限缩小至只与其行为直接相关的权限。

但授予细化的 URI 权限需要与拥有这些 URI 的内容提供程序进行一定的合作。强烈建议内容提供程序实施此功能，并且通过 [android:grantUriPermissions](https://developer.android.com/reference/android/R.styleable.html" \l "AndroidManifestProvider_grantUriPermissions) 属性或 [<grant-uri-permissions>](https://developer.android.com/reference/android/R.styleable.html#AndroidManifestGrantUriPermission) 标记声明支持此功能。

在 [Context.grantUriPermission()](https://developer.android.com/reference/android/content/Context.html" \l "grantUriPermission(java.lang.String, android.net.Uri, int))、[Context.revokeUriPermission()](https://developer.android.com/reference/android/content/Context.html" \l "revokeUriPermission(android.net.Uri, int)) 和 [Context.checkUriPermission()](https://developer.android.com/reference/android/content/Context.html" \l "checkUriPermission(android.net.Uri, int, int, int)) 方法中可以找到更多信息。

* **平台架构**



## Android Runtime

对于运行 Android 5.0（API 级别 21）或更高版本的设备，每个应用都在其自己的进程中运行，并且有其自己的 [Android Runtime (ART)](http://source.android.com/devices/tech/dalvik/index.html) 实例。ART 编写为通过执行 DEX 文件在低内存设备上运行多个虚拟机，DEX 文件是一种专为 Android 设计的字节码格式，经过优化，使用的内存很少。编译工具链（例如 [Jack](https://source.android.com/source/jack.html)）将 Java 源代码编译为 DEX 字节码，使其可在 Android 平台上运行。

ART 的部分主要功能包括：

* 预先 (AOT) 和即时 (JIT) 编译
* 优化的垃圾回收 (GC)
* 更好的调试支持，包括专用采样分析器、详细的诊断异常和崩溃报告，并且能够设置监视点以监控特定字段