Android 11
开发者手册

本手册包含哪些内容?

本手册旨在帮助您快速了解 Android 11。

在本手册中，您将了解到 Android 11 的新特性和 API、以及如何迁移您的应用到 Android 11。尤其已经对 Android 11 做过迁移的开发者，还会分享他们的最佳实践。我们期待这些分享会对您在 Android 11 应用的迁移方面有所帮助。手册还有一些 Android 11 开发的常见问题以及您可以参考的一些文字及视频资源。我们希望，这些内容可以帮助您快速将您的应用迁移到 Android 11，率先感受 Android 11 给您带来的各种创新和改善，以确保您的应用脱颖而出。

我们一直在不断寻找各种方法来改善我们的产品和服务，为此，我们也会定期更新本手册，以便为您提供最新信息。这意味着，当您使用本手册所描述的某些功能或特点时，他们可能会发生变化，在这种情况下，请遵循相应产品中的说明，或访问 Android 开发者官方网站。

目　录

如何阅读本手册

开发新手，可以从第一章开始阅读。内有 Android 11 的系统介绍。

经验高手，直接从第二章开始阅读，以便快速迁移现有应用。

带有以下标记符的标题，是提醒您重点关注的，标题下的内容供您快速浏览。

A B C

为方便开发者了解更多相关内容，本手册提供了大量扩展内容的网址（如：应用开发者文档，MediaStore，UsbManager API，系统提醒窗口变更，等），在 PDF 版本下可直接点击访问链接内容。

为方便开发者快速跳转到其他页面，各页面可通过点击页眉或按钮返回本小节首页或上一级章节首页，或者目录页。

为方便开发者通过手机上阅读相关内容，本手册最后一章提供了更多资源的二维码，开发者扫描相应的二维码即可进入页面。

第一章

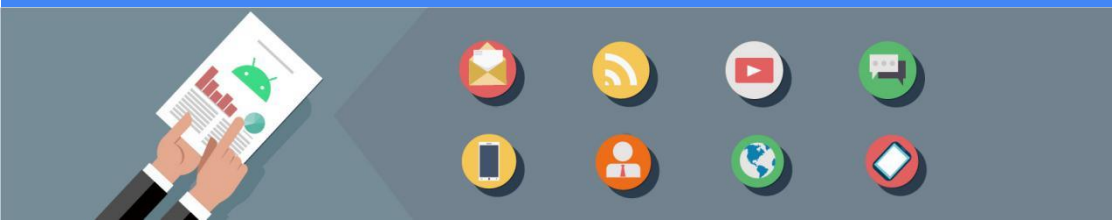
概述

目录

返回主目录

1.1 Android 操作系统概述

返回本章目录



Android，一个颠覆移动设备功能的平台。可为手机、平板电脑、手表、电视和汽车提供强力支持。它面向所有人开放，开发者、设计人员和设备制造商均可使用。能让更多人试验、畅想和创造前所未有的事物。它提供了开箱即用、持续运作的安全功能，致力于每天为超过 25 亿部使用中的 Android 设备增强安全性和隐私保护。在 Android 设备上，您可决定何时、是否分享自己的数据，您的隐私将由您随心掌控。我们通过开发各种无障碍功能和产品，让用户以各种方式体验精彩世界。

如果您想构建 Android 应用，建议您从 Kotlin 开始着手。Kotlin 是现代 Android 开发与指导性开发库的核心。世界各地的 Android 开发者向我们反馈称，Kotlin 极具表现力，能帮助大家编写更高质量的应用，而且可以轻松在现有的 Java 代码库中使用，因此都对 Kotlin 赞赏有加。在 Play Store 上排名前 1,000 的应用中，有 70% 以上现在使用 Kotlin。

1.1.1 可为手机、平板电脑、手表、电视和汽车提供强力支持

从只能让设备运行，到让生活更轻松，都是 Android 在背后提供强力支持。有了 Android，才能让 GPS 避开拥堵，用手表发短信，让 Google 助理回答问题。目前有 25 亿部活跃设备搭载了 Android 操作系统。Android 能够为各种设备提供强力支持，从 Google Pixel 4a、5G 手机到炫酷的平板电脑、手表、电视，不胜枚举。

可折叠手机搭载 Android 系统并能灵活调整屏幕开合角度来满足多种用途，让您可以像达人一样同时做多件事。多款搭载 Android 系统的手机在拍照方面达到世界一流水准。

Android One 手机已通过认证，可运行最简单、最新版本的 Android，每个月接收定期安全更新，享受至少两年的操作系统升级服务 2，并且仅搭载最基本的应用。

在 RAM 不到 2 GB 的智能手机上也可充分发挥 Android 系统的潜力。Google Play 也在不断上架专为 Android（Go 版本）开发的新应用。

只需几步，即可将文件、联系人信息和应用从一部手机，无论是 Android 手机，还是 iOS 手机，轻松转移到另一部 Android 手机。

1.1.2 孜孜不倦地推动实现一切可能

Android 面向所有人开放，开发者、设计人员和设备制造商均可使用。能让更多人试验、畅想和创造前所未有的事物。

快速翻译各种语言的内容，通过手机相机即可实现。在 Android 设备上，当您将镜头对准文字时，谷歌翻译可自动扫描文字，并将其转换成您想要的语言。该功能目前支持 25 种语言，即使离线也能使用。无论是菜单、路标，还是其他内容，只要您的相机可以拍下来，Android 就能进行翻译。

利用相机搜索网上内容。借助 Google 智能镜头，您只要将相机对准所见之物，即可执行相关搜索和其他操作。许多 Android 相机都内置了 Google 智能镜头。也就是说，您可以直接通过相机实时查询衣服和家装用品，复制和翻译文字，以及识别植物、动物和地标。

自动为手机上正在播放的媒体内容添加字幕。您只需点按一下，实时字幕功能就会自动为您的手机上正在播放的媒体内容添加字幕。该功能适用于任何应用中的视频、播客和语音消息，甚至是您自己录制的内容。

在 Android 设备上，您可决定何时、是否分享自己的数据，例如网络与应用活动记录、位置记录等。如果有应用在非使用期间访问您的位置信息，您将收到通知。此外，如果您想更改权限，可以在一个地方找到所有隐私设置。您的隐私将由您随心掌控。

您可使用各种“数字健康”工具了解手机使用情况，确定使用方

式。无论是休息片刻、排除干扰，还是睡前放松，“数字健康”都能帮到您。

每个人使用设备的方式不尽相同。为此，我们致力于开发各种无障碍功能和产品，例如屏幕阅读器、消音器，甚至 AR 步行导航。没有一种技术，适合所有人的需求。因此，Android 能让用户以各种方式体验精彩世界。

实时转写是一项新功能，可以捕捉人们所说的话并实时创建转录，支持 70 多种语言，并且会将您的数据保留在您的设备上，确保其不会外泄。该功能甚至会使用 Google 的语音识别软件来调整转录内容，以提高准确性。您只能在 Android 设备上使用该功能。

在 Android 设备上，您可以使用两种不同类型的屏幕阅读器。通过 TalkBack，您完全无需屏幕即可使用设备。或者，您可以点按通知或按钮等特定元素，让系统利用随选朗读功能读出这些元素。基本上只要是屏幕上的内容，Android 都可以读给您听。

声音增强器是一项由 Android 提供支持的无障碍功能，可以滤除噪音，放大声音的音量，并可进行微调，让您获享最佳听觉效果。无论您是在听讲座，看电视，还是在嘈杂的地方通话，只要插入耳机，便能听得更清楚。

1.1.3 内置安全功能，为您的手机持续护航

Android 为您提供了开箱即用、持续运作的安全功能。Google Play 保护机制扫描所有应用，软件定期安全更新，平台不断改进。它就像一位毫不懈怠的保安，不眠不休，时刻保障您的安全。

我们的目标是保护每位 Android 用户。我们致力于每天为超过 25 亿部使用中的 Android 设备增强安全性和隐私保护。

A 层层安全保护

Android 生态系统的各个环节相辅相成，共同致力于打造可顺畅高效运行的强力防护机制。Android 的层层防护机制会利用硬件和软件来防止设备遭到入侵，确保用户安全无虞。

Google Play 保护机制

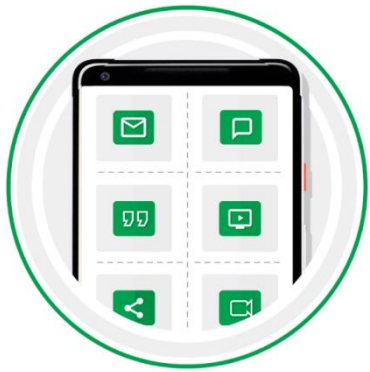
Google Play 保护机制是内置的恶意软件防护功能，可从应用层开始确保设备安全无虞。在 Google 机器学习技术的支持下，这套机制会不断调整和改进。它每天都会自动扫描 Android 手机上的所有应用，防止有害应用出现在手机上，这使得它成为全球部署最广泛的移动威胁防护服务。此外，内置

超过 1000 亿：每天扫描和验证的应用数

的“查找我的设备”功能可让设备即使丢失也能受到妥善保护。

内置安全保护措施的平台

除应用层之外，Android 平台还可以从内部协助确保设备安全无虞。应用沙盒可将每个 Android 应用隔离开并为其提供保护，以防止其他应用访问您的私人信息。我们还会防止他人非法访问操作系统内部组件，以免有人利用其中的错误进行攻击。此外，设备上的完整加密功能可确保数据安全无虞，即使设备不慎丢失也不必担心。



100%：已报告的 Android 平台重大安全漏洞，在 2019 年公开披露之前 Google 就发布了补丁程序

旨在提供防护的硬件

在最深层，我们会利用硬件本身来防止设备被非法访问。相关保护措施包括：安全锁定屏幕功能、确保设备未遭篡改的启动时验证功能，以及由硬件辅助的加密和密钥处理功能（以保障传输中及存储中的数据的安全）。

100%：运行 Nougat 或更高版本的 Android 手机使用基于硬件的锁屏验证

定期推出安全更新，确保所有保护机制正常运行

所有这些保护层都通过安全更新得到增强。我们每个月会定期为 Pixel 设备和加入 Android One 计划的设备提供更新。此外，我们还会通过全面的安全公告，将这些更新和漏洞的详情提供给我们的合作伙伴和用户。

超过 10 亿：设备在 2019 年收到安全更新

B 开放与透明

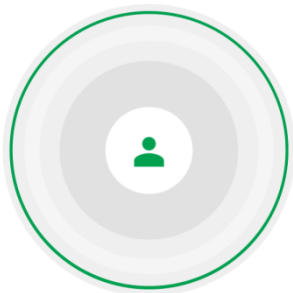
我们所做的一切都是公开透明的。无论是推出开源平台，还是为用户提供最新信息，我们都确保在整个社区分享相关知识。我们从开源平台入手，确保所做的更改公开透明。我们会随时向用户和合作伙伴社区成员通报最新信息，这可让 Android 社区中的所有人携起手来，共同努力提高安全性。

众人拾柴火焰高

我们的开源平台可以让成千上万的供应商、研究人员和设备制造商每天都能提供反馈、解决问题并为打造更安全的生态系统贡献力量。我们在 Android 安全奖

和 Google Play 安全奖计划上注入的资金超过了 690 万美元，旨在奖励那些花费时间和精力来发现漏洞的独立安全研究人员。

一切由用户掌控



我们的操作系统在设计上透明公开。如果某个应用尝试访问敏感数据，我们会要求该应用先获取用户授权。我们的权限请求和提醒都是公开透明的，用户能够得到所需的信息，充满自信地管理自己设备的安全设置。

C 由 Google 提供支持

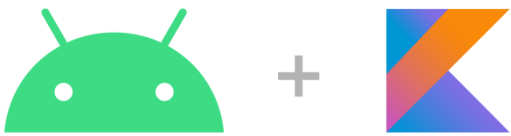
我们与 Google 的专家团队倾力合作，共同保障超过 25 亿部 Android 设备的安全。我们会与 Google 的各团队展开合作，为用户打造完善的端到端保护机制。举例来说，我们会与“身份管理”团队合作，一起确保帐号安全；还会与“安全浏览”团队合作，为应用和 Chrome 浏览提供保护。此外，我们还会与 Google AI 紧密合作，利用他们在人工智能和机器学习方面的专业知识来增强生态系统的安全性，以便实时为您提供保护。

1.1.4 在 Android 开发中优先采用 Kotlin

从最炙手可热的初创公司到财富 500 强公司，大量企业都采用 Kotlin 来构建其诸多应用。使用 Kotlin 更快地编写更出色的 Android 应用。 Kotlin 是一种新型的静态类型编程语言，有超过 60% 的专业 Android 开发者在使用，它有助于提高工作效率、开发者满意度和代码安全性。

在 2019 年 Google I/O 大会上，我们宣布今后将越来越优先采用 Kotlin 进行 Android 开发，并且也坚守了这一承诺。Kotlin 是一种富有表现力且简洁的编程语言，不仅可以减少常见代码错误，还可以轻松集成到现有应用中。如果您想构建 Android 应用，建议您从 Kotlin 开始着手，充分利用一流的 Kotlin 功能。

为了支持使用 Kotlin 进行 Android 开发，我们和另一组织联手创办了 Kotlin 基金会，不断投入人力物力来提高编译器性能和 build 速度。如需详细了解 Android 的 Kotlin 优先承诺，请参阅 Android 在 Kotlin 方面的承诺。



A

为什么要优先使用 Kotlin 进行开发?

我们查看了直接来自开发者、Google Developers 专家（GDE）的反馈，以及我们通过开发者调研获得的反馈。许多开发者已喜欢上使用 Kotlin，且提供更多 Kotlin 支持的呼声很高。下面介绍了开发者喜欢用 Kotlin 编写代码的原因：

- **富有表现力且简洁**：您可以使用更少的代码实现更多的功能。表达自己的想法，少编写样板代码。在使用 Kotlin 的专业开发者中，有 67% 的人反映其工作效率有所提高。
- **更安全的代码**：Kotlin 有许多语言功能，可帮助您避免 null 指针异常等常见编程错误。包含 Kotlin 代码的 Android 应用发生崩溃的可能性降低了 20%。
- **可互操作**：您可以在 Kotlin 代码中调用 Java 代码，或者在 Java 代码中调用 Kotlin 代码。Kotlin 可完全与 Java 编程语言互操作，因此您可以根据需要在项目中添加任意数量的 Kotlin 代码。
- **结构化并发**：Kotlin 协程让异步代码像阻塞代码一样易于使用。协程可大幅简化后台任务管理，例如网络调用、本地数据访问等任务的管理。

B

Kotlin 优先意味着什么？

在构建新的 Android 开发工具和内容（例如 Jetpack 库、示例、文档和培训内容）时，我们会在设计层面考虑到 Kotlin 用户，同时继续支持通过 Java 编程语言使用我们的 API。

	Java 语言	Kotlin
平台 SDK 支持	是	是
Android Studio 支持	是	是
Lint	是	是
引导式文档支持	是	是
API 文档支持	是	是
AndroidX 支持	是	是
AndroidX Kotlin 特有 API（KTX、协程等）	无	是
在线培训	尽力而为	是
示例	尽力而为	是
多平台项目	否	是
Jetpack Compose	否	是

C

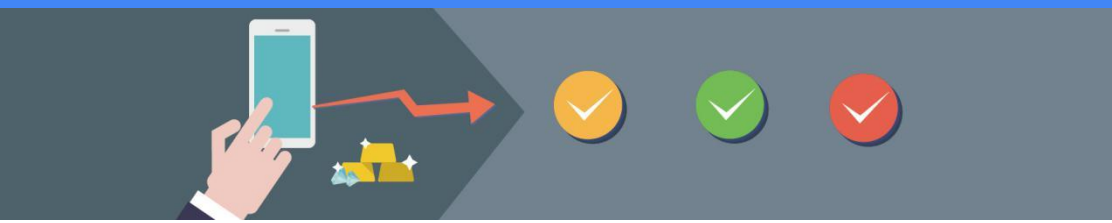
我们也使用 Kotlin!

如今，超过 60 款 Google 应用是用 Kotlin 构建的，其中包括 Google 地图、Google Home、Google Play、Google Pay 和 Google 云端硬盘等应用。我们的开发者喜欢 Kotlin 提供的语言功能。Google Home 团队改为采用 Kotlin 开发新功能后，其代码库大小缩减了 33%，且因出现 null 指针异常而导致应用崩溃的次数减少了 30%。

如需详细了解在 Android 开发中使用 Kotlin，请参阅在 Android 开发中使用 Kotlin 语言的常见问题解答。

1.2 Android 11 亮点

返回本章目录



Android 11 重点关注三个主题：以人为本的沟通方式、让用户快速访问和灵活控制所有智能设备，以及让用户有更多方式控制设备上的数据如何共享的隐私安全。

对于开发者来说，Android 11 带来了大量的新功能，包括会话通知、设备和媒体控制、单次权限、增强的 5G 支持、IME 切换效果等，欢迎大家积极尝试。为了帮助您更快地推进开发工作，我们还添加了新的工具，如兼容性开关、ADB 增量安装、应用退出原因 API、数据访问审核 API、Kotlin 可空性注解等。这些工作都是为了让开发者们能喜爱 Android 11！

下面，我们来介绍一下 Android 11 为开发者提供了哪些功能以及您现在能如何利用这些功能。

1.2.1 以人为本、灵活控制与隐私安全

以人为本

Android 11 致力于凸显人的要素，且善于沟通。我们重塑了您在手机上进行沟通的方式，也让操作系统能识别出那些对您来说更重要的人，让您能更快速地和他们联系。对于开发者来说，Android 11 可以帮助您在应用中实现更深入的会话和更个性化的互动体验。

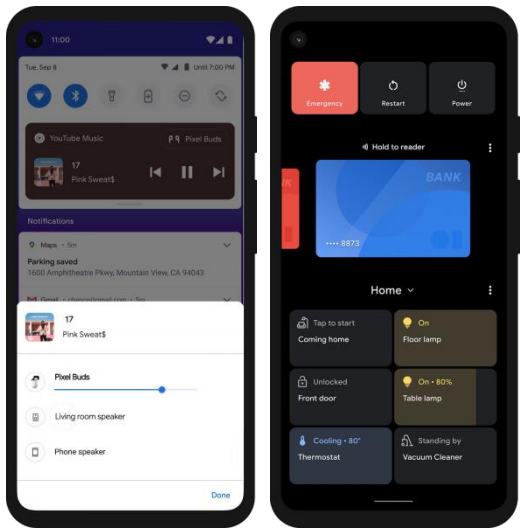
- **会话通知**会显示在通知栏顶部的专门区域，其设计更凸显联系对象，且提供了会话特定的操作，例如以 Bubbles 的形式打开聊天、在主屏幕中创建会话快捷方式，以及设置提醒。
- Bubbles 可以让用户在手机上进行多任务切换时依然保持对话可见并且可交互。消息和聊天应用可以通过基于通知的 Bubbles API，在 Android 11 上提供这种全新体验。
- **键盘提示整合功能**可以让自动填写应用以及 IME（输入法编辑器）在 IME 建议栏中安全地向用户提供基于上下文的实体和字符串，使得输入更加便利。

B 灵活控制

Android 11 让用户们得以快速访问所有的智能设备，并集中控制它们。开发者们则可以通过全新的 API 来帮助用户控制智能设备和管理媒体播放：

- 设备控制（Device Controls）
  让用户得以更快、更轻松地访问和控制他们连接的设备。只需长按电源按钮就可以调出设备控制菜单，一站式完成设备控制。应用也可以通过新的 API 出现在这个控制菜单中。详细信息请访问官方文档。

- 媒体控制（Media Controls）
  让用户得以更快捷地切换音频和视频内容的播放设备——不论是耳机、麦克风还是电视。详细信息请访问官方文档。



设备控制和媒体控制

C

隐私安全

在 Android 11 中，我们为用户带来了更高的掌控能力，让他们能更好地管理敏感权限。我们还会通过更快速的更新来持续确保设备安全。

单次授权 - 用户现在可以授予应用一次性的权限来访问设备的麦克风、摄像头或者位置信息。应用下次被使用时需要再次请求权限。详细信息请访问官方文档。

Android 11 中的单次授权对话框

后台位置 - 想访问后台位置信息现在需要用户在授予运行时权限外进行更进一步的操作。如果您的应用需要访问后台位置信息，系统会要求您必须先请求前台位置权限。您可以通过单独的权限申请来进一步要求访问后台位置信息，系统会将用户带到设置页面（Settings）中完成授权操作。

另外需要注意的是，我们在今年二月宣布，Google Play 开发者需要获得批准后才可以让应用在后台访问位置信息，以防止滥用。现在我们为开发者提供更长的时间来做出修改，在 2021 年之前我们不会强行要求现有的应用遵守本政策。详细信息请访问官方文档。

权限自动重置 - 如果用户在很长一段时间里未使用某应用，Android 11 将 "自动重置" 所有与该应用关联的运行时权限并通知用户。在用户下次使用该应用时，应用可以再次请求权限。详细信息请访问官方文档。

分区存储 - 我们一直在努力更好地保护外部存储上的应用和用户数据，还加入了更多的改进以便让开发者更轻松地进行迁移。详细信息请访问官方文档。

Google Play 系统更新 - 自去年发布以来，Google Play 系统更新让我们能更快速地更新操作系统核心组件，并覆盖 Android 生态系统中的众多设备。在 Android 11 中，可更新的模块数量增加了一倍有余，新增的 12 个可更新模块，为用户和开发者带来更好的隐私性、安全性和一致性。

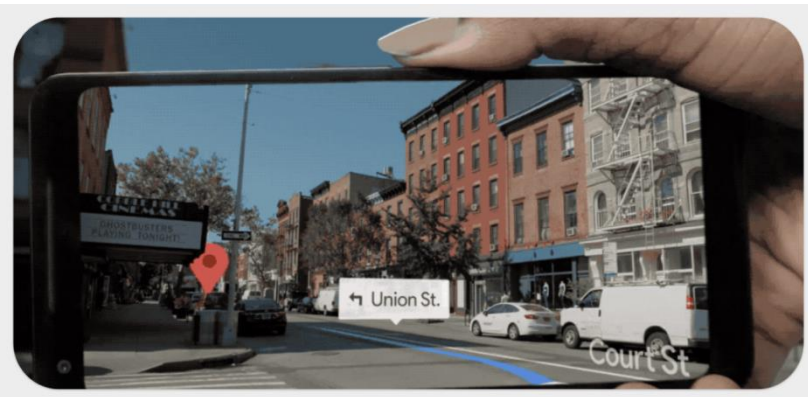
BiometricPrompt API - 开发者现在可以通过 BiometricPrompt API 来指定其应用所需的生物识别身份验证强度类型，用来解锁或者访问应用中的敏感内容。为了向下兼容，我们也将这些功能加入到了 Jetpack Biometric 开发库中。随着工作的进展，我们会为大家带来进一步的更新。

身份认证 API（Identity Credential API） - 这个 API 会带来全新的使用场景，支持包括驾驶执照、国民身份证和数字身份证。我们正在与各政府机构和行业伙伴合作，以确保 Android 11 为数字化身份认证体验做好准备。

在这里阅读有关 Android 11 隐私功能的详细信息。

1.2.2 实用创新

更强的 5G 支持 - Android 11 可以让开发者利用 5G 网络更快的速度和更低的延迟。您可知晓用户何时连接到 5G 网络，查看连接是否处于计费状态，并且估测连接的带宽。为了帮助您即刻打造 5G 体验，我们也在 Android Emulator 中加入了 5G 支持。请访问 5G 开发者网页，了解如何在 Android 上使用 5G 功能。



将高速体验带出家门，5G 可以让您的随行移动体验更加流畅，让您随时与周边环境、朋友、家人互动并满足工作的需要

新的屏幕类型 - 设备厂商们也在持续进行创新，将新的屏幕形态投入市场，包括挖孔屏和瀑布屏。Android 11 已经在平台中增加了对这些屏幕的支持，并提供了相应的 API 方便您优化应用。您可以通过现有的 Display Cutout API 来管理挖孔屏和瀑布屏。您可以通过设置新的窗口布局属性来使用整个瀑布屏，

并通过瀑布屏边衬区（insets）API 来管理屏幕边缘附近的互动。

呼叫过滤服务 - Android 11 可以帮助呼叫过滤应用更好地管理骚扰电话。应用在呼叫详细信息中可以获取来电的 STIR/SHAKEN 验证状态（这个标准可以防止来电 ID 欺诈），并能报告拒接来电的原因。应用还可以自定义系统提供的呼叫后屏幕（post call screen），方便用户执行诸如 "将呼叫方标记为骚扰电话" 或 "添加到联系人" 之类的操作。

1.2.3 优化与品质

操作系统弹性 - 在 Android 11 中，我们通过对内存回收操作(比如根据 RSS HWM 阈值强制用户无法感知的进程重启）进行微调，使操作系统整体更具动态性和弹性。另外，为了改善性能和内存的使用，Android 11 还增加了 Binder 缓存，通过缓存那些检索相对静态数据的系统服务，优化了使用率高的 IPC 调用。Binder 缓存还通过减少 CPU 时间延长了电池寿命。

同步 IME 切换效果 - 这是一组全新的 API，让您可以在 IME（输入法编辑器，也叫软键盘）和系统栏进出屏幕时同步调整应用中的内容，从而更轻松地创建出自然、直观、流畅的 IME 切换效果。为了确保切换时做到逐帧精确，新的 WindowInsetsAnimation.Callback API 会在系统栏或 IME 移动时逐帧告知应用边衬区的变化。此外，您可以通过新的 WindowInsetsAnimationController API 控制系统 UI，包括系统栏、IME、沉浸模式等。阅读这篇博文了解更多。

HEIF 动画可绘制对象 - ImageDecoder API 现在允许您解码和渲染存储在 HEIF 文件中的图像序列动画，方便您引入高品质的素材，同时最大程度地减少流量消耗和 APK 尺寸。相对于 GIF 动画，HEIF 图像序列可以显著减小文件尺寸。

原生图像解码器 - 应用可以使用新的 NDK API 来通过原生代码解码和编码图像（如 JPEG、PNG、WebP），以便进行图形或后期处理，而且因为您无需捆绑外部代码库，从而得以保持

较小的 APK 尺寸。原生解码器还可以从 Android 持续的平台安全更新中获益。我们提供了 NDK 样例代码作为使用参考。

MediaCodec 中的低延迟视频解码 - 低延迟视频对于 Stadia 等实时视频流应用和服务至关重要。支持低延迟播放的视频编解码器会在解码开始后尽快返回流的第一帧。应用可以使用新 API 来针对特定编解码器检查和配置低延迟播放。

可变刷新率 - 应用和游戏现在可以通过新的 API 为其窗口设置首选帧率。大多数 Android 设备以 60Hz 的刷新率更新屏幕，但是某些设备支持多种刷新率，例如 90Hz 和 60Hz，并可在运行时切换。在这些设备上，系统会基于首选帧率来为应用选择最佳刷新率。您可以通过 SDK 和 NDK 来使用该 API。详细信息请访问官方文档。

动态资源加载器 - Android 11 提供了一个新的公开 API 来让应用在运行时动态加载资源和素材。通过 Resource Loader 框架，您可以在应用或游戏中包含一套基本资源，然后在运行时根据需要加载其他资源，或更改已加载的资源。

Neural Networks API (NNAPI) 1.3 - 我们持续增加算子和控制，以支持 Android 设备上的机器学习。为了优化常见的使用场景，NNAPI 1.3 增加了优先级和超时、内存域（memory domains）以及异步指令队列的 API。新的算子支持包含有符号整数非对称量化以及分支和循环的高级模型，hard-swish 算子则可以用于加速下一代设备上视觉模型（如 MobileNetV3）。

1.2.4 开发者体验

应用兼容性工具 - 我们努力将大多数 Android 11 行为变更设置为可选择开启，从而最大限度地减少对兼容性带来的影响，除非您将应用的 targetSdkVersion 设置为 30，否则这些变更不会生效。如果您是通过 Google Play 发布应用，则有一年多的时间来选择支持这些变更，但我们建议尽早开始测试。为了帮助您进行测试，Android 11 允许您单独开启或关闭其中的许多变更。详细信息请访问官方文档。

应用退出原因 - 了解应用退出的原因以及当时的状态十分重要——包括应用所在的设备类型、内存配置和运行场景。Android 11 通过退出原因 API 让这个事情变得更加容易：您可以使用该 API 来查看应用最近退出的详细信息。

数据访问审核 - 数据访问审核可以让您更好地了解自己的应用访问用户数据的情况，以及访问来自的用户流程。例如，它能帮您识别无意的私有数据访问，不论其来自于您自己的代码还是其他 SDK。详细信息请访问官方文档。

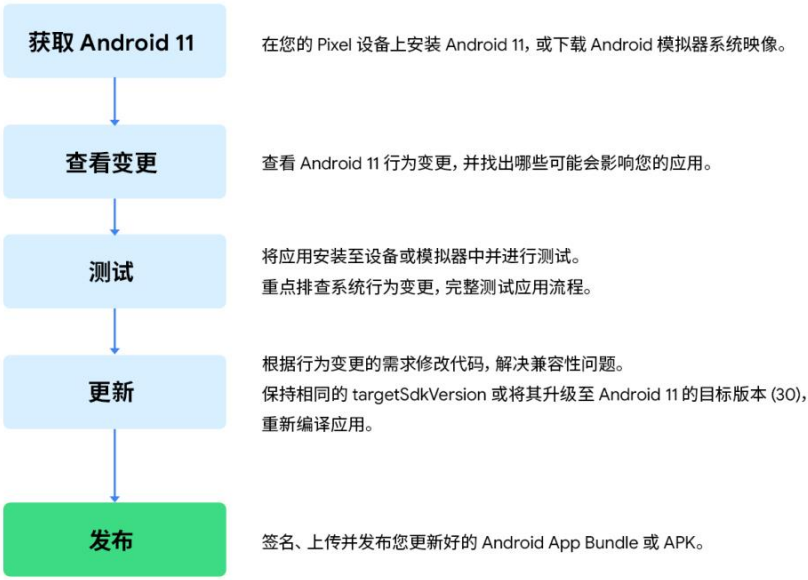
ADB 增量安装（**ADB Incremental**）- 在开发过程中使用 ADB（Android Debug Bridge）安装体积较大的 APK 可能会拖慢速度，影响您的工作效率，对 Android 游戏开发者而言尤其如此。Android 11 带来了 ADB Incremental，现在从开发机向 Android 11 设备上部署大型 APK（2GB 以上）的速度可以提高 10 倍之多。详细信息请访问官方文档。

Kotlin 可空性注解 - Android 11 为公共 API 中的更多方法

增

加了可空性注解。而且，它将一些现有的注解从警告升级为错误。这可以帮助您在构建时就发现问题，不用等到运行时才出错。阅读此文了解更多。

1.2.5 让您的应用为 Android 11 做好准备



请首先关注针对所有应用的行为变更：

下面是首先需要关注的行为变更（无论您应用的 targetSdkVersion 是多少）：

- **单次权限** - 现在，用户可以为位置信息、设备麦克风和摄像头授予单次使用权限。详细信息请访问官方文档。
- **外部存储访问权限** - 应用无法再访问外部存储空间中其他应用的文件。详细信息请访问官方文档。

- Scudo Hardened Allocator - 现在它是应用内原生代码的堆内存分配器。详细信息请访问官方文档。
- **文件描述符排查器** - 此功能现在默认启用，以检测应用原生代码的文件描述符处理错误。详细信息请访问官方文档。

Android 11 中还有许多可选择支持的行为变更 - 您的应用如果针对新平台发布，才会受到影响。我们建议在您发布应用的兼容版本后尽快评估这些变更。有关兼容性测试和工具的更多信息，请查看 Android 11 兼容性相关的资源，并访问 Android 11 开发者网站了解技术细节。

1.2.6 使用新功能和 API 改进您的应用

准备就绪后，请深入研究 Android 11 并了解您可以使用的新功能和 API。下面是一些您可以优先考虑的重点功能。

我们推荐所有应用支持这些功能:

- **深色主题**（自 Android 10 开始支持）- 通过添加 Dark Theme（深色主题）或启用 Force Dark，确保为启用全系统深色主题的用户提供一致的体验。
- **手势导航**(自 Android 10 开始支持)- 请支持手势导航，包括提供边到边的沉浸式体验，以及确保自定义手势与默认手势配合良好。详细信息请访问官方文档。
- **共享快捷方式**（自 Android 10 开始支持）- 想要接收共享数据的应用应该使用共享快捷方式 API 来创建共享目标。想要发送共享数据的应用应确保使用 Android Sharesheet。
- **同步 IME 切换效果** - 使用新的 WindowInsets 和相关 API 为用户提供流畅的切换效果。详细信息请阅读这篇博文。
- **新的屏幕类型** - 对挖孔屏或瀑布屏设备，请确保根据需要针对这些屏幕测试和调整您的内容。详细信息请访问官方文档。

我们还推荐这些功能，如果它们和您的应用体验契合的话:

- **会话** - 消息和通信应用可以通过提供长效共享快捷方式和在通知中呈现对会话来融入用户的对话体验。详细信息请访问官方文档。
- 聊天气泡（Bubbles）- Bubbles 可以在多任务切换时依然保持对话可见及可用。应用通过基于通知的 Bubbles API 来实现此功能。
- 5G - 如果您的应用或内容可以利用 5G 更快的速度和更低的延迟，请参考我们的开发者资源，开始构建 5G 体验。
- **设备控制** - 如果您的应用支持外部智能设备，请确保这些设备可以从新的 Android 11 设备控制菜单访问。详细信息请访问官方文档。
- **媒体控制** - 对于媒体应用，我们建议支持 Android 11 媒体控制，这样用户就可以从快速设置（Quick Settings）菜单中管理媒体播放。详细信息请访问官方文档。

您可以前往 https://developer.android.google.cn/about/versions/11 了解更多有关 Android 11 功能的信息。

1.3

Android 11 中的隐私权

返回本章目录

1.3.1 概览

Android 11 基于 Android 早期版本构建，增加了多种功能和更新，以保障用户安全并提高透明度和可控性。所有开发者都应查看隐私功能并测试他们的应用。具体影响可能会因每个应用的核心功能、目标平台和其他因素而异。

如需详细了解 Android 11 中生效的主要变更，请参阅以下部分。

重大隐私权变更

	隐私权变更	受影响的应用	缓解策略
√	强制执行分区存储机制 以 Android 11 或更高版本为目标平台的应用始终会受分区存储行为的影响	以 Android 11 或更高版本为目标平台的应用，以及以 Android 10 为目标平台且未将 requestLegacyExternalStorage 设为 true 以停用分区存储的应用	更新您的应用以使用分区存储 详细了解分区存储变更
√	单次授权 使用单次授权功能，用户可以授予对位置信息、麦克风和摄像头的临时访问权限	在 Android 11 或更高版本上运行且请求位置信息、麦克风或摄像头权限的应用	在尝试访问受某项权限保护的数据之前，检查您的应用是否具有该权限 遵循请求权限方面的最佳做法

√	自动重置权限 如果用户在 Android 11 或更高版本上几个月未与应用互动，系统会自动重置应用的敏感权限	以 Android 11 或更高版本为目标平台且在后台执行大部分工作的应用	要求用户阻止系统重置应用的权限 详细了解自动重置权限
√	后台位置信息访问权限 Android 11 更改了用户向应用授予后台位置信息权限的方式	以 Android 11 或更高版本为目标平台且需要在后台访问位置信息的应用	通过对权限请求方法的多次单独调用，逐步请求在前台（粗略或精确）和后台访问位置信息的权限。必要时，说明用户授予该权限所能得到的益处 详细了解 Android 11 中的在后台访问位置信息的权限
√	软件包可见性 Android 11 更改了应用查询同一设备上的其他已安装应用及与之互动的方式	以 Android 11 或更高版本为目标平台且与设备上的其他已安装应用交互的应用	将 <queries> 元素添加到应用的清单 详细了解软件包可见性
√	前台服务 Android 11 更改了前台服务访问位置信息、摄像头和麦克风相关数据的方式	在 Android 11 或更高版本上运行且在前台服务中访问位置信息、摄像头或麦克风的应用	分别针对需要访问摄像头和麦克风的前台服务，声明 camera 和 microphone 前台服务类型。但请注意，应用在后台运行时启动的前台服务通常无法访问位置信息、摄像头或麦克风。 详细了解前台服务的变更

开始使用隐私权更新

1. **查看隐私功能：**评估应用。查看应用如何存储文件和用户数据、请求权限以及请求位置信息。此外，查看应用与其他应用互动的方式，考虑对应用访问的数据进行审核，并确定是否需要更新使用前台服务的方式。

2. **在 Android 11 上测试您的应用：**在 Android 11 上运行应用。使用应用兼容性工具评估各项系统变更对您的应用的影响。

3. **更新应用：**如有可能，以 Android 11 为目标平台，让用户参与测试并发布更新。

1.3.2 存储机制更新

Android 11（API 级别 30）进一步增强了平台功能，为外部存储设备上的应用和用户数据提供了更好的保护。此版本引入了多项增强功能，例如，可主动选择启用的媒体原始文件路径访问机制、面向媒体的批量编辑操作，以及存储访问框架的界面更新。

此版本还改进了分区存储，以便开发者更轻松地迁移到此存储模型。如需了解详情，请参阅 Android 存储用例和最佳做法指南，以及 Android 11 存储常见问题解答文章。

强制执行分区存储

在 Android 11 上运行但以 Android 10（API 级别 29）为目标平台的应用仍可请求 requestLegacyExternalStorage 属性。应用可以利用此标记暂时停用与分区存储相关的变更，例如授予对不同目录和不同类型的媒体文件的访问权限。当您将应用更新为以 Android 11 为目标平台后，系统会忽略 requestLegacyExternalStorage 标记。

保持与 Android 10 的兼容性

如果应用在 Android 10 设备上运行时选择退出分区存储，建议您继续在应用的清单文件中将 requestLegacyExternalStorage 设为 true。这样，应用就可以在运行 Android 10 的设备上继续按预期运行。

将数据迁移到使用分区存储时可见的目录

如果您的应用使用旧版存储模型且之前以 Android 10 或更低版本为目标平台，您可能会将数据存储到启用分区存储模型后您的应用无法访问的目录中。在以 Android 11 为目标平台之前，请将数据迁移到与分区存储兼容的目录。

测试分区存储

如需在您的应用中启用分区存储，而不考虑应用的目标 SDK 版本和清单标记值，请启用以下应用兼容性标记：

- DEFAULT_SCOPED_STORAGE（默认情况下，对所有应用处于启用状态）
- FORCE_ENABLE_SCOPED_STORAGE（默认情况下，对所有应用处于停用状态）

如需**停用**分区存储而改用旧版存储模型，请**取消设置**这两个标记。

B 管理设备存储空间

从 Android 11 开始，使用分区存储模型的应用只能访问自身的应用专用缓存文件。如果您的应用需要管理设备存储空间，请按照关于如何查询可用空间的说明操作。

1. 通过调用 ACTION_MANAGE_STORAGE intent 操作检查可用空间。
2. 如果设备上的可用空间不足，请提示用户同意让您的应用清除所有缓存。为此，请调用 ACTION_CLEAR_APP_CACHE intent 操作。

> ⓘ 注意：ACTION_CLEAR_APP_CACHE intent 操作会严重影响设备的电池续航时间，并且可能会从设备上移除大量的文件。

C 外部存储设备上的应用专用目录

从 Android 11 开始，应用无法在外部存储设备上创建自己的应用专用目录。如需访问系统为您的应用提供的目录，请调用 getExternalFilesDirs()。

D 媒体文件访问权限

为了在保证用户隐私的同时可以更轻松地访问媒体，Android 11 增加了以下功能。

执行批量操作

为实现各种设备之间的一致性并增加用户便利性，Android 11 添加了多种方法，以便开发者更轻松地管理媒体文件组。

使用直接文件路径和原生库访问文件

为了帮助您的应用更顺畅地使用第三方媒体库，Android 11 允许您使用除 MediaStore API 之外的 API 通过直接文件路径访问共享存储空间中的媒体文件。其中包括：

- File API。
- 原生库，例如 fopen()。

E 访问其他应用中的数据

为保护用户的隐私，在搭载 Android 11 或更高版本的设备上，系统会进一步对您的应用访问其他应用的私有目录的行为进行限制。

访问内部存储设备上的数据目录

Android 9（API 级别 28）开始限制哪些应用可使其内部存储设备上数据目录中的文件可由其他应用进行全局访问。以 Android 9 或

更高版本为目标平台的应用不能使其数据目录中的文件全局可访问。

Android 11 在此限制的基础上进行了扩展。如果您的应用以 Android 11 为目标平台，则不能访问其他任何应用的数据目录中的文件，即使其他应用以 Android 8.1（API 级别 27）或更低版本为目标平台且已使其数据目录中的文件全局可读也是如此。

访问外部存储设备上的应用专用目录

在 Android 11 上，应用无法再访问外部存储设备中的任何其他应用的专用于特定应用的目录中的文件。

F 文档访问限制

为让开发者有时间进行测试，以下与存储访问框架 (SAF) 相关的变更只有在应用以 Android 11 或更高版本为目标平台时才会生效。

访问目录

您无法再使用 `ACTION_OPEN_DOCUMENT_TREE` intent 操作请求访问以下目录：

- 内部存储卷的根目录。
- 设备制造商认为可靠的各个 SD 卡卷的根目录，无论该卡是模拟卡还是可移除的卡。可靠的卷是指应用在大多数情况下可以成功访问的卷。
- `Download` 目录。

访问文件

您无法再使用 ACTION_OPEN_DOCUMENT_TREE 或 ACTION_OPEN_DOCUMENT intent 操作请求用户从以下目录中选择单独的文件：

- Android/data/ 目录及其所有子目录。
- Android/obb/ 目录及其所有子目录。

测试变更

如需测试此行为更改，请执行以下操作：

1. 通过 ACTION_OPEN_DOCUMENT 操作调用 intent。检查 Android/data/ 和 Android/obb/ 目录是否均不显示。
2. 执行以下某项操作：
   - 启用 RESTRICT_STORAGE_ACCESS_FRAMEWORK 应用兼容性标记。
   - 以 Android 11 或更高版本为目标平台。
3. 通过 ACTION_OPEN_DOCUMENT_TREE 操作调用 intent。检查 Download 目录是否已显示，以及与目录关联的操作按钮是否呈灰显状态。

G 权限

Android 11 引入了与存储权限相关的以下变更。

以任何版本为目标平台

不管应用的目标 SDK 版本是什么，以下变更均会在 Android 11 中生效：

- **存储**运行时权限已重命名为**文件和媒体**。
- 如果您的应用未停用分区存储并且请求 READ_EXTERNAL_STORAGE 权限，用户会看到不同于 Android 10 的对话框。该对话框表明您的应用正在请求访问照片和媒体，如图 1 所示。

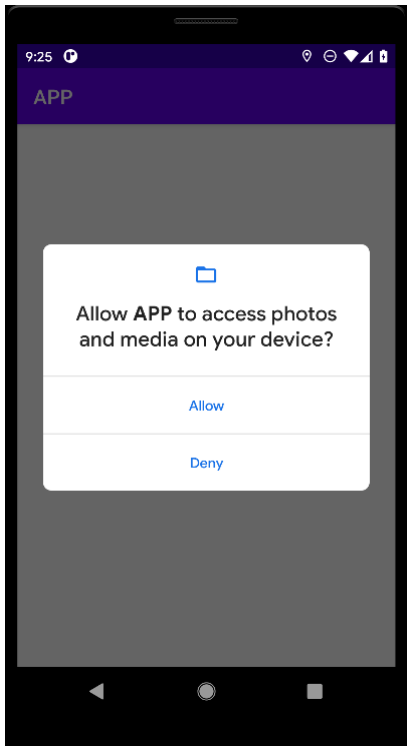


图 1. 应用使用分区存储并请求 READ_EXTERNAL_STORAGE 权限时显示的对话框。

用户可以在系统设置中查看哪些应用具有 READ_EXTERNAL_STORAGE 权限。在**设置** > **隐私** > **权限管理器** > **文件和媒体**页面上，具有该

权限的每个应用都列在**允许存储所有文件**下。如果您的应用以 Android 11 为目标平台，请记住，对“所有文件”的这种访问权限是只读访问权限。如需使用此应用读取**和写入**共享的存储空间中的所有文件，需要具有所有文件访问权限。

以 Android 11 为目标平台

如果应用以 Android 11 为目标平台，那么 WRITE_EXTERNAL_STORAGE 权限和 WRITE_MEDIA_STORAGE 特许权限将不再提供任何其他访问权限。

请注意，在搭载 Android 10（API 级别 29）或更高版本的设备上，您的应用可以提供明确定义的媒体集合，例如 MediaStore.Downloads，而无需请求任何存储相关权限。详细了解如何在处理应用中的媒体文件时仅请求必要的权限。

H 所有文件访问权限

绝大多数需要共享存储空间访问权限的应用都可以遵循共享媒体文件和共享非媒体文件方面的最佳做法。但是，某些应用的核心用例需要广泛访问设备上的文件，但无法采用注重隐私保护的存储最佳做法高效地完成这些操作。对于这些情况，Android 提供了一种名为“所有文件访问权限”的特殊应用访问权限。如需了解详情，请参阅有关如何管理存储设备上的所有文件的指南。

1.3.3 权限更新

在 Android 11 中，用户能够针对位置信息、麦克风和摄像头指定更精细的权限。此外，如果以 Android 11 或更高版本为目标平台的应用在一段时间内未使用，系统就会重置这些应用的权限。如果应用使用系统提醒窗口或读取与电话号码相关的信息，可能需要更新它们声明的权限。

单次授权

从 Android 11 开始，每当应用请求与位置信息、麦克风或摄像头相关的权限时，面向用户的权限对话框会包含**仅限这一次**选项。如果用户在对话框中选择此选项，系统会向应用授予临时的单次授权。

详细了解系统如何处理单次授权。

注意：如果应用在请求运行时权限时已遵循最佳做法，您无需更改应用即可支持单次授权。

B 自动重置未使用的应用的权限

如果应用以 Android 11 或更高版本为目标平台并且数月未使用，系统会通过自动重置用户已授予应用的运行时敏感权限来保护用户数据。此操作与用户在系统设置中查看权限并将应用的访问权限级别更改为**拒绝**的做法效果一样。如果应用遵循了有关在运行时请求权限的最佳做法，那么您不必对应用进行任何更改。这是因为，当用户与应用中的功能互动时，您应该会验证相关功能是否具有所需权限。

> ★ **注意**：系统仅重置运行时权限，在请求这些权限时，系统会向用户显示运行时提示。

详细了解系统如何自动重置未使用的应用的权限。

C 权限对话框的可见性

从 Android 11 开始，在应用安装到设备上后，如果用户在使用过程中多次针对某项特定的权限点按**拒绝**，那么在您的应用再次请求该权限时，用户将不会看到系统权限对话框。该操作表示用户希望“不再询问”。在之前的版本中，除非用户先前已选中“不再询问”对话框或选项，否则每当您的应用请求权限时，用户都会看到系统权限对话框。Android 11 中的这一行为变更旨在避免重复请求用户已选择拒绝的权限。

> ★ **注意**：如果您的应用已遵循与权限相关的最佳做法，您无需更改您的应用即可支持此行为变更。

详细了解如何在应用中处理权限请求遭拒情况。

D 系统提醒窗口变更

在 Android 11 中，向应用授予 SYSTEM_ALERT_WINDOW 权限的方式发生了一些变更。这些变更可以让权限的授予更有目的性，从而达到保护用户的目的。

根据请求自动向某些应用授予 SYSTEM_ALERT_WINDOW 权限

系统会根据请求自动向某些类型的应用授予 SYSTEM_ALERT_WINDOW 权限：

- 系统会自动向具有 ROLE_CALL_SCREENING 且请求 SYSTEM_ALERT_WINDOW 的所有应用授予该权限。如果应用失去 ROLE_CALL_SCREENING，就会失去该权限。
- 系统会自动向通过 MediaProjection 截取屏幕且请求 SYSTEM_ALERT_WINDOW 的所有应用授予该权限，除非用户已明确拒绝向应用授予该权限。当应用停止截取屏幕时，就会失去该权限。此用例主要用于游戏直播应用。

这些应用无需发送 ACTION_MANAGE_OVERLAY_PERMISSION 以获取 SYSTEM_ALERT_WINDOW 权限，它们只需直接请求 SYSTEM_ALERT_WINDOW 即可。

MANAGE_OVERLAY_PERMISSION intent 始终会将用户转至系统权限屏幕

从 Android 11 开始，ACTION_MANAGE_OVERLAY_PERMISSION intent 始终会将用户转至顶级**设置**屏幕，用户可在其中授予或撤消应用的 SYSTEM_ALERT_WINDOW 权限。intent 中的任何 package: 数据都会被忽略。

在更低版本的 Android 中，ACTION_MANAGE_OVERLAY_PERMISSION intent 可以指定一个软件包，它会将用户转至应用专用屏幕以管理权限。从 Android 11 开始将不再支持此功能，而是必须由用户先选择要授予或撤消哪些应用的权限。此变更可以让权限的授予更有目的性，从而达到保护用户的目的。

E 电话号码

Android 11 更改了您的应用在读取电话号码时使用的与电话相关的权限。

如果您的应用以 Android 11 或更高版本为目标平台，并且需要访问以下列表中显示的电话号码 API，则必须请求 READ_PHONE_NUMBERS 权限，而不是 READ_PHONE_STATE 权限。

- TelephonyManager 类和 TelecomManager 类中的 getLine1Number() 方法。
- TelephonyManager 类中不受支持的 getMsisdn() 方法。

如果您的应用声明 READ_PHONE_STATE 以调用前面列表中的方法以外的方法，您可以继续在所有 Android 版本中请求 READ_PHONE_STATE。不过，如果您仅对前面列表中的方法使用 READ_PHONE_STATE 权限，请按以下方式更新您的清单文件：

1. 更改 READ_PHONE_STATE 的声明，以使您的应用仅在 Android 10（API 级别 29）及更低版本中使用该权限。
2. 添加 READ_PHONE_NUMBERS 权限。

以下清单声明代码段演示了此过程：

```
<manifest>
    <!-- Grants the READ_PHONE_STATE permission only on devices that
run  Android 10 (API level 29) and lower. -->
    <uses-permission android:name="READ_PHONE_STATE"
android:maxSdkVersion="29" />
    <uses-permission android:name="READ_PHONE_NUMBERS" />
</manifest>
```

1.3.4 位置信息更新

为了进一步保护用户隐私，Android 11 增加了单次位置信息访问权限，并更改了用户授予在后台访问位置信息权限的方式。这些更新会影响到 Android 11 及更高版本上运行的所有应用。

A 单次访问权限

在 Android 11 及更高版本中，每当应用请求在前台访问位置信息时，系统权限对话框都包含一个名为**仅限这一次**的选项，如图 1 所示。通过这一选项，用户可以更好地控制应用何时有权访问位置信息。

详细了解系统如何处理单次授权。

*图 1. 用于授予前台位置权限的系统对话框，其中包含一个名为**仅限这一次**的选项。*

B

在后台访问位置信息的权限

Android 11 更改了应用中的功能获取后台位置信息访问权限的方式。本部分介绍了上述各项变更。

如果应用中的某项功能从后台访问位置信息，请验证此类访问是否有必要，并考虑以其他方式获取该功能所需的信息。如需详细了解在后台访问位置信息的权限，请参阅在后台访问位置信息页面。

单独请求在后台访问位置信息

正如有关如何在运行时请求位置信息访问权限的指南中所述，您应该执行递增位置信息请求。如果您的应用以 Android 11 或更高版本为目标平台，系统会强制执行此最佳做法。如果您同时请求在前台访问位置信息的权限和在后台访问位置信息的权限，系统会忽略该请求，且不会向您的应用授予其中的任一权限。

权限对话框的变更

变更详情
变更名称：BACKGROUND_RATIONALE_CHANGE_ID 变更 ID：147316723
如何切换

在测试应用与 Android 11 的兼容性时，您可以使用以下 ADB 命令开启或关闭此变更：
$ adb shell am compat enable (*147316723|BACKGROUND_RATIONALE_CHANGE_ID*) *PACKAGE_NAME*
$ adb shell am compat disable (*147316723|BACKGROUND_RATIONALE_CHANGE_ID*) *PACKAGE_NAME*
如需详细了解兼容性框架以及如何切换变更的状态，请参阅测试应用与 Android 11 的兼容性。

在搭载 Android 11 或更高版本的设备上，您的应用中的某项功能请求在后台访问位置信息时，系统对话框不会包含用于启用在后台访问位置信息权限的按钮。如需启用在后台访问位置信息的权限，用户必须在设置页面上针对应用的位置权限设置**一律允许**选项，如介绍如何请求在后台访问位置信息的指南中所述。

1.3.5 软件包可见性

Android 11 更改了应用查询用户已在设备上安装的其他应用以及与之交互的方式。使用 <queries> 元素，应用可以定义一组自身可访问的其他软件包。通过告知系统应向您的应用显示哪些其他软件包，此元素有助于鼓励最小权限原则。此外，此元素还可帮助 Google Play 等应用商店评估应用为用户提供的隐私权和安全性。

如果您的应用以 Android 11 或更高版本为目标平台，您可能需要在应用的清单文件中添加 <queries> 元素。在 <queries> 元素中，您可以按软件包名称、intent 签名或提供程序授权指定软件包。

注意：即使您的应用以 Android 11 或更高版本为目标平台，某些应用也会自动对您的应用可见。如需详细了解您应用的用例如何受到影响以及如何更新您的应用，请参阅有关如何根据用例配置软件包可见性的指南。

如需了解详情，请参阅有关如何管理其他已安装应用的软件包可见性（您的应用以 Android 11 或更高版本为目标平台时）。

A

测试变更

如需测试此行为变更是否已在您的应用中生效，请完成以下步骤：

1. 安装 Android Studio 3.6.1 或更高版本。
2. 安装 Android Studio 支持的最新版 Gradle。
3. 将应用的 targetSdkVersion 设为 30。
4. 不要在应用的清单文件中添加 <queries> 元素。
5. 调用 getInstalledApplications() 或 getInstalledPackages()。这两种方法都应返回过滤后的列表。
6. 查看应用的哪些功能无法正常使用。
7. 引入适当的 <queries> 条目来修复这些功能。

1.3.6 数据访问审核

为了让应用及其依赖项访问用户私密数据的过程更加透明，Android 11 引入了数据访问审核功能。借助此流程得出的见解，您可以更好地识别可能出现的意外数据访问。您的应用可以注册 AppOpsManager.OnOpNotedCallback 实例，该实例可在每次发生以下任一事件时执行相应操作：

- 应用的代码访问私密数据。为了帮助您确定应用的哪个逻辑部分调用了事件，您可以按归因标记审核数据访问。
- 依赖库或 SDK 中的代码访问私密数据。

如需了解详情，请参阅有关如何审核对数据的访问权限的指南。

1.3.7 MAC 地址更新

Android 11 引入了与 MAC 地址相关的变更。这些变更只会影响以 Android 11 为目标平台的应用。如需详细了解这些变更，请参阅 Android 11。

1.3.8 前台服务

在 Android 11 中，前台服务何时可以访问设备的位置信息、摄像头和麦克风发生了一些变化。这有助于保护敏感的用户数据。

A 前台服务类型 camera 和 microphone

如果您的应用以 Android 11 或更高版本为目标平台，且在前台服务中访问摄像头或麦克风，则必须添加前台服务类型 `camera` 和 `microphone`。

B 对在使用时访问的限制

如果您的应用在后台运行时启动了某项前台服务，则该前台服务无法访问麦克风或摄像头。此外，除非您的应用具有在后台访问位置信息的权限，否则该服务无法访问位置信息。

详细了解如何在应用中使用前台服务。

1.4

影响应用的行为变更

返回本章目录

1.4.1 行为变更：所有应用

Android 11 平台包含一些行为变更，这些变更可能会影响您的应用。以下行为变更将影响在 Android 11 上运行的所有应用，无论其采用哪种 targetSdkVersion 都不例外。您应该测试您的应用，然后根据需要进行修改，以适当地支持这些变更（如果适用）。

此外，请务必查看仅影响以 Android 11 为目标平台的应用的行为变更列表。

隐私设置

Android 11 引入了一些变更和限制来加强用户隐私保护，其中包括：

- 单次授权：让用户可以选择授予更多对位置信息、麦克风和摄像头的临时访问权限。
- 权限对话框的可见性：一再拒绝某项权限表示用户希望“不再询问”。
- 数据访问审核：深入了解您的应用在何处访问私密数据，无论是在您的应用自己的代码中，还是在依赖库的代码中。
- 系统提醒窗口权限：根据请求自动向某些类型的应用授予 SYSTEM_ALERT_WINDOW 权限。此外，包含

ACTION_MANAGE_OVERLAY_PERMISSION intent 操作的 intent 始终会将用户转至系统设置中的屏幕。

- 永久 SIM 卡标识符：在 Android 11 及更高版本中，使用 getIccId() 方法访问不可重置的 ICCID 受到限制。该方法会返回一个非 null 的空字符串。如需唯一标识设备上安装的 SIM 卡，请改用 getSubscriptionId() 方法。订阅 ID 会提供一个索引值（从 1 开始），用于唯一识别已安装的 SIM 卡（包括实体 SIM 卡和电子 SIM 卡）。除非设备恢复出厂设置，否则此标识符的值对于给定 SIM 卡是保持不变的。

如需了解详情，请参阅隐私设置页面。

B 接触史通知

Android 11 在更新平台时考虑了接触史通知系统。用户现已可在 Android 11 上运行接触史通知应用，且无需开启设备位置信息设置。接触史通知系统的设计使得使用该系统的应用无法通过蓝牙扫描推断设备所处的位置，因此，此例外情况仅适用于接触史通知系统。

为保护用户的隐私，所有其他应用仍无法执行蓝牙扫描，除非用户已开启设备位置信息设置且已授予相应应用位置权限。如需了解详情，请阅读我们的接触史通知最新动态博文。

C

安全性

SSL 套接字默认情况下使用 Conscrypt SSL 引擎

Android 的默认 SSLSocket 实现基于 Conscrypt。从 Android 11 开始，该实现是在 Conscrypt 的 SSLEngine 之上内部构建的。

Scudo Hardened Allocator

Android 11 在内部使用 Scudo Hardened Allocator 为堆分配提供服务。Scudo 能够检测并减轻某些类型的内存安全违规行为。如果您在原生代码崩溃报告中发现与 Scudo 相关的崩溃（例如 Scudo ERROR:），请参阅 Scudo 问题排查文档。

应用使用情况统计信息

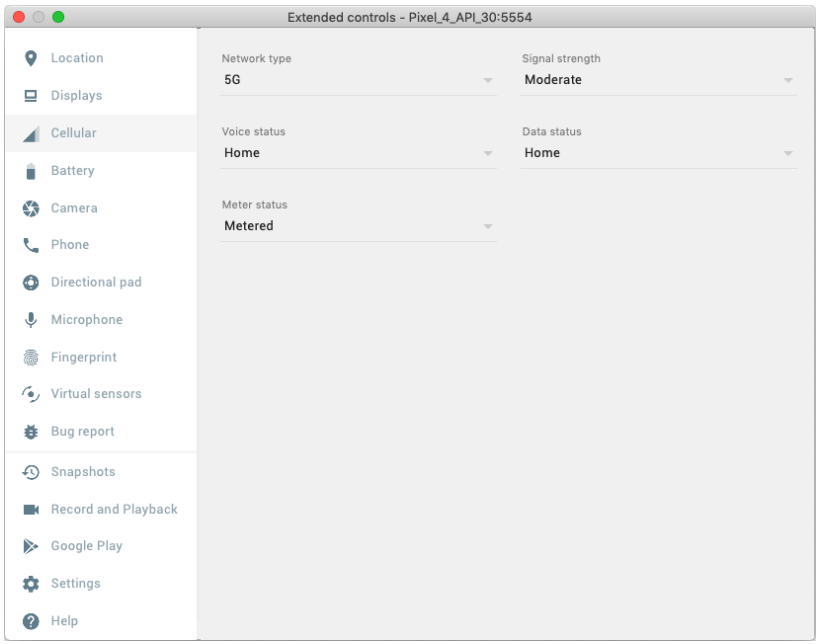
为了更好地保护用户，Android 11 将每个用户的应用使用情况统计信息存储在凭据加密存储空间中。因此，系统和任何应用都无法访问该数据，除非 isUserUnlocked() 返回 true，这发生在出现以下某种情况之后：

- 用户在系统启动后首次解锁其设备。
- 用户在设备上切换到自己的帐号。

如果您的应用已绑定到 UsageStatsManager 的实例，请检查您是否是在用户解锁其设备后在此对象上调用方法。如果并非如此，该 API 现在会返回 null 或空值。

针对 5G 的模拟器支持

Android 11 添加了 5G API，使您的应用能够添加各种先进的功能。如需在添加这些功能时对其进行测试，您可以使用 Android SDK 模拟器的新功能。这项新功能是在模拟器版本 30.0.22 中添加的。选择 5G 网络设置可将 TelephonyDisplayInfo 设为 OVERRIDE_NETWORK_TYPE_NR_NSA，修改带宽估算值，还允许您设置按流量计费性，以验证您的应用是否会对 NET_CAPABILITY_TEMPORARILY_NOT_METERED 状态的变化做出适当的响应。



D

性能和调试

JobScheduler API 调用限制调试

Android 11 为应用提供调试支持，以便确定有可能超过特定速率限制的 JobScheduler API 调用。开发者可以利用此服务发现潜在的性能问题。对于 debuggable 清单属性设置为 true 的应用，超出速率限制的 JobScheduler API 调用将返回 RESULT_FAILURE。如此设置限制，正当合理的用例应该就不会受到影响。

文件描述符排错程序 (fdsan)

Android 10 引入了 fdsan（文件描述符排错程序）。fdsan 检测错误处理文件描述符所有权的错误，例如 use-after-close 和 double-close。在 Android 11 中，fdsan 的默认模式发生了变化。现在，fdsan 会在检测到错误时中止，而以前的行为则是记录警告并继续。如果您在应用中发现由于 fdsan 而导致的崩溃，请参阅 fdsan documentation。

E

非 SDK 接口限制

Android 11 包含更新后的受限制非 SDK 接口列表（基于与 Android 开发者之间的协作以及最新的内部测试）。在限制使用非 SDK 接口之前，我们会尽可能确保有可用的公开替代方案。

如果您的应用并非以 Android 11 为目标平台，那么其中一些变更可能不会立即对您产生影响。不过，虽然您目前可以使用一些非 SDK 接口（具体取决于应用的目标 API 级别），但只要您使用任何非 SDK 方法或字段，应用无法运行的风险终归较高。

如果您不确定自己的应用是否使用了非 SDK 接口，则可以测试该应用，进行确认。如果您的应用依赖于非 SDK 接口，您应该开始计划迁移到 SDK 替代方案。然而，我们知道某些应用具有使用非 SDK 接口的有效用例。如果您无法为应用中的某项功能找到使用非 SDK 接口的替代方案，则应请求新的公共 API。

如需详细了解此 Android 版本中的变更，请参阅 Android 11 中有关限制非 SDK 接口的更新。如需全面了解有关非 SDK 接口的详细信息，请参阅对非 SDK 接口的限制。

F V1 版 Google 地图共享库已移除

Android 11 中已完全移除 V1 版 Google 地图共享库。此库之前已被弃用，并已停止在 Android 10 中的应用中运行。对于搭载 Android 9（API 级别 28）或更低版本的设备，之前依赖于此共享库的应用应改用适用于 Android 的 Google 地图 SDK。

> ★ **注意：**迁移到 Maps SDK for Android 后，请务必从应用清单文件的 <uses-library> 元素中移除对 v1 版 Google 地图共享库的引用。应用无法再将 Google Play 过滤与 V1 版 Google 地图共享库和 <uses-library> 元素一起使用。

G 与其他应用交互

分享内容 URI

如果您的应用与其他应用分享内容 URI，相应 intent 必须至少设置以下 intent 标记中的一个，以便授予对 URI 的访问权限：FLAG_GRANT_READ_URI_PERMISSION 和 FLAG_GRANT_WRITE_URI_PERMISSION。这样一来，如果其他应用以 Android 11 为目标平台，相应应用仍可访问内容 URI。即使内容 URI 与不属于您的应用的内容提供程序相关联，您的应用也必须包含 intent 标记。

如果您的应用拥有已与内容 URI 相关联的内容提供程序，请确认该内容提供程序未被导出。我们已建议采用这项安全最佳做法。

1.4.1 行为变更：所有应用

1.4.2 行为变更：以 Android 11 为目标平台的应用

与早期版本一样，Android 11 包含一些行为变更，这些变更可能会影响您的应用。以下行为变更仅影响以 Android 11 或更高版本为目标平台的应用。如果您的应用将 targetSdkVersion 设为 30，您应酌情修改自己的应用，以便正确支持这些行为。

此外，请务必查看对 Android 11 上运行的所有应用都有影响的行为变更列表。

隐私设置

Android 11 引入了一些变更和限制来加强用户隐私保护，其中包括：

- 强制执行分区存储：对外部存储目录的访问仅限于应用专用目录，以及应用已创建的特定类型的媒体。
- 自动重置权限：如果用户几个月未与应用互动，系统会自动重置应用的敏感权限。
- 在后台访问位置信息的权限：用户必须转到系统设置，才能向应用授予在后台访问位置信息的权限。
- 软件包可见性：当应用查询设备上已安装应用的列表时，系统会过滤返回的列表。

如需了解详情，请参阅隐私设置页面。

B 安全

堆指针标记

变更详情
变更名称：NATIVE_HEAP_POINTER_TAGGING 变更 ID：135754954
如何切换

在测试应用与 Android 11 的兼容性时，您可以使用以下 ADB 命令开启或关闭此变更：
$ adb shell am compat enable (*135754954*|*NATIVE_HEAP_POINTER_TAGGING*) *PACKAGE_NAME*
$ adb shell am compat disable (*135754954*|*NATIVE_HEAP_POINTER_TAGGING*) *PACKAGE_NAME*
如需详细了解兼容性框架以及如何切换变更的状态，请参阅测试应用与 Android 11 的兼容性。

现在，堆指针在最高有效字节 (MSB) 中有一个非零标记。错误地使用指针的应用（包括修改 MSB 的应用）现在会崩溃或遇到其他问题。这是支持未来启用了 ARM 内存标记扩展 (MTE) 的硬件所必需的变更。如需了解详情，请参阅已加标记的指针。

如需停用此功能，请参阅 allowNativeHeapPointerTagging 清单文档。

消息框的更新

来自后台的自定义消息框被屏蔽

出于安全方面的考虑，同时也为了保持良好的用户体验，如果包含自定义视图的消息框是以 Android 11 或更高版本为目标平台的应用从后台发送的，系统会屏蔽这些消息框。请注意，仍允许使用文本消息框；此类消息框是使用 Toast.makeText() 创建的，并不调用 setView()。

如果您的应用仍尝试从后台发布包含自定义视图的消息框，系统不会向用户显示相应的消息，而是会在 logcat 中记录以下消息：

```
W/NotificationService: Blocking custom toast from package \
<package> due to package not in the foreground
```

消息框回调

如果您希望在消息框（文本消息框或自定义消息框）出现或消失时收到通知，请使用 Android 11 中添加的 addCallback() 方法。

文本消息框 API 变更

以 Android 11 或更高版本为目标平台的应用会发现文本消息框受到以下负面影响：

- getView() 方法返回 null。

- 以下方法的返回值并不反映实际值，因此您不应在应用中依赖于它们：
    - `getHorizontalMargin()`
    - `getVerticalMargin()`
    - `getGravity()`
    - `getXOffset()`
    - `getYOffset()`
- 以下方法是空操作，因此您的应用不应使用它们：
    - `setMargin()`
    - `setGravity()`

C

网络连接

限制对 APN 数据库的读取访问

变更详情
变更名称：APN_READING_PERMISSION_CHANGE_ID 变更 ID：124107808
如何切换

在测试应用与 Android 11 的兼容性时，您可以使用以下 ADB 命令开启或关闭此变更：
$ adb shell am compat enable (*124107808*|*APN_READING_PERMISSION_CHANGE_ID*) *PACKAGE_NAME*
$ adb shell am compat disable (*124107808*|*APN_READING_PERMISSION_CHANGE_ID*) *PACKAGE_NAME*
如需详细了解兼容性框架以及如何切换变更的状态，请参阅测试应用与 Android 11 的兼容性。

以 Android 11 为目标平台的应用现在必须具备 Manifest.permission.WRITE_APN_SETTINGS 特权，才能读取或访问电话提

供程序 APN 数据库。如果在不具备此权限的情况下尝试访问 APN 数据库，会生成安全异常。

D 无障碍服务

在清单文件中声明与 TTS 引擎的交互

由于软件包可见性发生了变更，因此以 Android 11 为目标平台且与文字转语音 (TTS) 引擎交互的应用需要将以下 <queries> 元素添加到其清单文件中：

```
<queries>
  <intent>
    <action
      android:name="android.intent.action.TTS_SERVICE" />
  </intent>
</queries>
```

在元数据文件中声明“无障碍”按钮使用情况

变更详情
变更名称：REQUEST_ACCESSIBILITY_BUTTON_CHANGE 变更 ID：136293963

如何切换

在测试应用与 Android 11 的兼容性时，您可以使用以下 ADB 命令开启或关闭此变更：

```
$ adb shell am compat enable
(136293963|REQUEST_ACCESSIBILITY_BUTTON_CHANGE)
PACKAGE_NAME

$ adb shell am compat disable
(136293963|REQUEST_ACCESSIBILITY_BUTTON_CHANGE)
PACKAGE_NAME
```

如需详细了解兼容性框架以及如何切换变更的状态，请参阅测试应用与 Android 11 的兼容性。

从 Android 11 开始，您的无障碍服务无法在运行时声明与系统的“无障碍”按钮相关联。如果您将 AccessibilityServiceInfo.FLAG_REQUEST_ACCESSIBILITY_BUTTON 附加到 AccessibilityServiceInfo 对象的 flags 属性，框架就不会将“无障碍”按钮回调事件传递给您的服务。

如需在无障碍服务中收到无障碍回调事件，请使用无障碍服务元数据文件声明您的服务与“无障碍”按钮的关联。在 accessibilityFlags 属性的定义中添加 flagRequestAccessibilityButton 值。无障碍服务元数据文件的常用位置为 res/raw/accessibilityservice.xml。

E

相机

媒体 intent 操作需要系统默认相机

从 Android 11 开始，只有预装的系统相机应用可以响应以下 intent 操作：

- android.media.action.VIDEO_CAPTURE
- android.media.action.IMAGE_CAPTURE
- android.media.action.IMAGE_CAPTURE_SECURE

如果有多个预装的系统相机应用可用，系统会显示一个对话框，供用户选择应用。如果您希望自己的应用使用特定的第三方相机应用来代表其捕获图片或视频，可以通过为 intent 设置软件包名称或组件来使这些 intent 变得明确。

F

应用打包和安装

压缩的资源文件

变更详情
变更名称：RESOURCES_ARSC_COMPRESSED 变更 ID：132742131
如何切换

在测试应用与 Android 11 的兼容性时，您可以使用以下 ADB 命令开启或关闭此变更：
$ adb shell am compat enable (*132742131*|*RESOURCES_ARSC_COMPRESSED*) *PACKAGE_NAME*
$ adb shell am compat disable (*132742131*|*RESOURCES_ARSC_COMPRESSED*) *PACKAGE_NAME*
如需详细了解兼容性框架以及如何切换变更的状态，请参阅测试应用与 Android 11 的兼容性。

如果以 Android 11（API 级别 30）或更高版本为目标平台的应用包含压缩的 resources.arsc 文件或者如果此文件未按 4 字节边界对齐，应用将无法安装。如果存在其中任意一种情况，系统将无法对此文件进行内存映射。无法进行内存映射的资源表必须读入 RAM 中的缓冲区，从而给系统造成不必要的内存压力，并大大增加设备的 RAM 使用量。

现在需要 APK 签名方案 v2

对于以 Android 11 (API 级别 30) 为目标平台，且目前仅使用 APK 签名方案 v1 签名的应用，现在还必须使用 APK 签名方案 v2 或更高版本进行签名。用户无法在搭载 Android 11 的设备上安装或更新仅通过 APK 签名方案 v1 签名的应用。

如需验证您的应用是否已使用 APK 签名方案 v2 或更高版本进行签名，您可以在命令行中使用 Android Studio 或 apksigner 工具。

> **注意**：为支持运行旧版 Android 的设备，除了使用 APK 签名方案 v2 或更高版本为您的 APK 签名之外，您还应继续使用 APK 签名方案 v1 进行签名。

G Firebase

Firebase JobDispatcher 和 GCMNetworkManager

如果您的应用以 API 级别 30 或更高级别为目标平台，在搭载 Android 6.0（API 级别 23）或更高版本的设备上会停用 Firebase JobDispatcher 和 GcmNetworkManager API 调用。如需了解迁

移，请参阅从 Firebase JobDispatcher 迁移到 WorkManager 和从 GCMNetworkManager 迁移到 WorkManager。

H 设备到设备文件传输

如果您的应用以 Android 11 或更高版本为目标平台，您将无法使用 allowBackup 属性停用应用文件的设备到设备迁移。系统会自动启用此功能。

不过，即使您的应用以 Android 11 或更高版本为目标平台，您也可以通过将 allowBackup 属性设为 false 来停用应用文件的云端备份和恢复。

I OnSharedPreferenceChangeListener 的回调变更

变更详情
变更名称：CALLBACK_ON_CLEAR_CHANGE 变更 ID：119147584
如何切换

在测试应用与 Android 11 的兼容性时，您可以使用以下 ADB 命令开启或关闭此变更：
`$ adb shell am compat enable (119147584|CALLBACK_ON_CLEAR_CHANGE) PACKAGE_NAME`
`$ adb shell am compat disable (119147584|CALLBACK_ON_CLEAR_CHANGE) PACKAGE_NAME`
如需详细了解兼容性框架以及如何切换变更的状态，请参阅测试应用与 Android 11 的兼容性。

对于以 Android 11（API 级别 30）为目标平台的应用，现在每次调用 Editor.clear 时，都会使用 null 键回调 OnSharedPreference

ChangeListener.onSharedPreferenceChanged。

J 非 SDK 接口限制

Android 11 包含更新后的受限制非 SDK 接口列表（基于与 Android 开发者之间的协作以及最新的内部测试）。在限制使用非 SDK 接口之前，我们会尽可能确保有可用的公开替代方案。

如果您的应用并非以 Android 11 为目标平台，那么其中一些变更可能不会立即对您产生影响。不过，虽然您目前可以使用一些非 SDK 接口（具体取决于应用的目标 API 级别），但只要您使用任何非 SDK 方法或字段，应用无法运行的风险终归较高。

如果您不确定自己的应用是否使用了非 SDK 接口，则可以测试该应用，进行确认。如果您的应用依赖于非 SDK 接口，您应该开始计划迁移到 SDK 替代方案。然而，我们知道某些应用具有使用非 SDK 接口的有效用例。如果您无法为应用中的功能找到无需使用非 SDK 接口的替代方案，则应请求添加新的公共 API。

如需详细了解此 Android 版本中的变更，请参阅 Android 11 中有关限制非 SDK 接口的更新。如需全面了解有关非 SDK 接口的详细信息，请参阅对非 SDK 接口的限制。

1.4.3 Android 11 中有关限制非 SDK 接口的更新

Android 11 包含更新后的受限制非 SDK 接口列表（基于与 Android 开发者之间的协作以及最新的内部测试）。在限制使用非 SDK 接口之前，我们会尽可能确保有可用的公开替代方案。

如果您的应用并非以 Android 11 为目标平台，那么其中一些变更可能不会立即对您产生影响。不过，虽然您目前可以使用一些非 SDK 接口（具体取决于应用的目标 API 级别），但只要您使用任何非 SDK 方法或字段，应用无法运行的风险终归较高。

如果您不确定自己的应用是否使用了非 SDK 接口，则可以测试该应用，进行确认。如果您的应用依赖于非 SDK 接口，您应该开始计划迁移到 SDK 替代方案。然而，我们知道某些应用具有使用非 SDK 接口的有效用例。如果您无法为应用中的某项功能找到使用非 SDK 接口的替代方案，应请求新的公共 API。

A 非 SDK 测试 API 现在受到限制

从 Android 11 开始，默认情况下，非 SDK 测试 API（即 AOSP 中带有 @TestApi 注释的 API）现在被屏蔽。这些非 SDK 接口用于在 Android 平台上执行内部测试。应用可以继续使用在其目标 API 级别上不受限制的非 SDK 测试 API，但任何新的测试 API 都会被列入屏蔽名单。

B Android 11 的列表变更

Android 11 中的列表变更分为以下几类：

- 在 Android 10（API 级别 29）中不受支持（列入了灰名单）而目前在 Android 11（API 级别 30）中被屏蔽的非 SDK 接口。
- 在 Android 11 中被添加到 Android SDK 的非 SDK 接口。

如需查看 Android 11（API 级别 30）的所有非 SDK 接口的完整列表，请下载以下文件：hiddenapi-flags.csv。

目前在 Android 11 中被屏蔽的非 SDK 接口

变更详情
变更名称：HIDE_MAXTARGETSDK_Q_HIDDEN_APIS 变更 ID：149994052
如何切换
在测试应用与 Android 11 的兼容性时，您可以使用以下 ADB 命令开启或关闭此变更： $ adb shell am compat enable (*149994052*|*HIDE_MAXTARGETSDK_Q_HIDDEN_APIS*) *PACKAGE_NAME*

```
$ adb shell am compat disable
(149994052|HIDE_MAXTARGETSDK_Q_HIDDEN_APIS)
PACKAGE_NAME
```

如需详细了解兼容性框架以及如何切换变更的状态，请参阅测试应用与 Android 11 的兼容性。

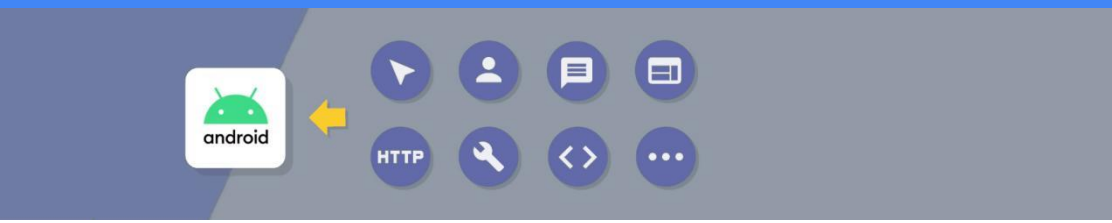
以下代码框列出了在 Android 10（API 级别 29）中不受支持（列入了灰名单）而在 Android 11（API 级别 30）中被屏蔽的所有非 SDK 接口。也就是说，这些接口属于 max-target-q (greylist-max-q) 列表，因此只有在您的应用以 Android 10（API 级别 29）或更低版本为目标平台时，才能使用这些接口。

我们的目标是在限制使用非 SDK 接口之前确保有可用的公开替代方案，并且我们知道您的应用可能具有使用这些接口的有效用例。如果您的应用在先前版本中使用的某个接口在 Android 11 中被屏蔽，则应针对该接口请求新的公共 API。

1.5

Android 11 功能和 API

返回本章目录



Android 11 面向开发者引入了一些出色的新功能和 API。以下几部分内容可帮助您了解适用于您的应用的功能并开始使用相关 API。

有关新增、修改和移除的 API 的详细列表，请参阅 API 差异报告。如需详细了解新的 API，请访问 Android API 参考文档 - 新 API 会突出显示以方便查看。此外，如需了解平台变更可能会在哪些方面影响您的应用，请务必查看会影响以 Android R 为目标平台的应用和所有应用的 Android 11 行为变更，以及隐私权变更。

1.5.1 概览

Android 11 的新体验以及在隐私权、安全、性能和质量及其他功能等方面引入的新功能和 API，详见以下各节摘要说明。

新体验

设备控件

Android 11 包含一个新的 `ControlsProviderService` API，可用于提供所连接的外部设备的控件。这些控件显示于 Android 电源菜单中的**设备控制器**下。如需了解详情，请参阅控制外部设备。

媒体控件

Android 11 更新了媒体控件的显示方式。媒体控件显示于快捷设置旁。来自多个应用的会话排列在一个可滑动的轮播界面中，其中包括在手机本地播放的会话流、远程会话流（例如在外部设备上检测到的会话或投射会话）以及可继续播放的以前的会话（按上次播放的顺序排列）。

用户无需启动相关应用即可在轮播界面中重新开始播放以前的会话。当播放开始后，用户可按常规方式与媒体控件互动。

如需了解详情，请参阅媒体控件。

屏幕

更好地支持瀑布屏

Android 11 提供了一些 API 以支持瀑布屏，这是一种无边框的全面屏。这种显示屏被视为刘海屏的变体。现有的 DisplayCutout.getSafeInset…() 方法现在会返回能够避开瀑布区域以及刘海的安全边衬区。如需在瀑布区域中呈现您的应用内容，请执行以下操作：

- 调用 DisplayCutout.getWaterfallInsets() 以获取瀑布边衬区的精确尺寸。
- 将窗口布局属性 layoutInDisplayCutoutMode 设为 LAYOUT_IN_DISPLAY_CUTOUT_MODE_ALWAYS，以允许窗口延伸到屏幕各个边缘上的刘海和瀑布区域。您必须确保刘海或瀑布区域中没有重要的内容。

> ★ 注意：如果您未将上述窗口布局属性设为 LAYOUT_IN_DISPLAY_CUTOUT_MODE_ALWAYS，Android 会在黑边模式下显示窗口，从而避开缺口和瀑布区域。

合页角度传感器和可折叠设备

使用 Android 11，可以通过以下方法使运行在采用合页式屏幕配置的设备上的应用能够确定合页角度：提供具有 TYPE_HINGE_ANGLE 的新传感器，以及新的 SensorEvent，后者可以监控合页角度，并提供设备的两部分之间的角度测量值。您可以使用这些原始测量值在用户操作设备时执行精细的动画显示。

请参阅可折叠设备。

对话

改进了会话

Android 11 对会话的处理方式进行了多项改进。会话是两人或更多人之间的实时双向通信。这些会话具有特殊的重要性，并且用户在如何与其进行交互方面有多个新的选项可以选择。

如需详细了解对话以及您的应用如何支持对话，请参阅人与对话。

聊天气泡

现已面向开发者推出气泡功能，该功能有助于在系统中显示会话。对话泡是 Android 10 中的一项实验性功能，通过开发者选项启用；在 Android 11 中，这项功能不再是必选功能。

如果应用以 Android 11（API 级别 30）或更高版本为目标平台，除非其通知满足新的对话要求，否则不会以 Android 10 对话泡形式显示。具体而言，通知必须与快捷方式关联。

在 Android 11 之前，如果您希望通知以气泡形式显示，需要明确指定将其设为始终在文档界面模式下启动。从 Android 11 开始，您不再需要明确进行这项设置；如果通知以对话泡形式显示，平台会自动将其设为始终在文档界面模式下启动。

对话泡功能有多项改进，用户可以更灵活地在每个应用中启用和停用对话泡功能。对于实现了实验性支持的开发者，Android 11 中的 API 有一些变更：

- 不带参数的 BubbleMetadata.Builder() 构造函数已弃用。请改为使用 BubbleMetadata.Builder(PendingIntent, Icon) 或

BubbleMetadata.Builder(String) 这两个新构造函数中的任意一个。

- 通过调用 BubbleMetadata.Builder(String)，根据快捷方式 ID 创建 BubbleMetadata。传递的字符串应与提供给 Notification.Builder 的快捷方式 ID 一致。
- 使用 Icon.createWithContentUri() 或新方法 createWithAdaptiveBitmapContentUri() 创建气泡图标。

5G 图标显示

如需了解如何在用户的设备上显示 5G 图标，请参阅在用户连接到 5G 网络时显示相关信息。

B 隐私权

Android 11 引入了大量变更和限制，目的是加强用户隐私保护。如需了解详情，请参阅隐私权页面。

C

安全

生物识别身份验证机制更新

为了帮助您控制应用数据的安全级别，Android 11 对生物识别身份验证机制进行了多项改进。这些变更也会在 Jetpack Biometric 库中显示。

身份验证类型

Android 11 引入了 BiometricManager.Authenticators 接口，可用于声明您的应用支持的身份验证类型。

确定用户所用的身份验证类型

在用户进行身份验证后，您可以通过调用 getAuthenticationType() 检查用户是使用设备凭据还是生物识别凭据进行的身份验证。

对“每次使用时进行身份验证”密钥的额外支持

Android 11 提供了对“每次使用时进行身份验证”密钥的更多支持。

已弃用的方法

Android 11 弃用了以下方法：

- setDeviceCredentialAllowed() 方法。
- setUserAuthenticationValidityDurationSeconds() 方法。
- 不带任何参数的 canAuthenticate() 过载版本。

安全共享大型数据集

在某些情况下，例如涉及机器学习或媒体播放时，您的应用可能需要与其他应用使用同一个大型数据集。在较早的 Android 版本中，您的应用与其他应用需要各自单独下载该数据集。

为帮助减少网络中和磁盘上的数据冗余，Android 11 允许使用共享数据 blob 在设备上缓存这些大型数据集。如需详细了解如何共享数据集，请参阅有关共享大型数据集的深度指南。

因 OTA 更新而重启设备后在未提供用户凭据的情况下执行文件级加密

设备完成 OTA 更新并重启后，放在受凭据保护的存储空间中的凭据加密 (CE) 密钥可立即用于执行文件级加密 (FBE) 操作。这意味着，完成 OTA 更新后，您的应用可以在用户输入其 PIN 码、解锁图案或密码之前恢复需要 CE 密钥的操作。

★ **注意：**此变更仅影响因 OTA 更新而发生的设备重启。如果您的应用在设备重启后用户输入其 PIN 码、解锁图案或密码之前始终需要访问 CE 密钥，请继续支持直接启动。

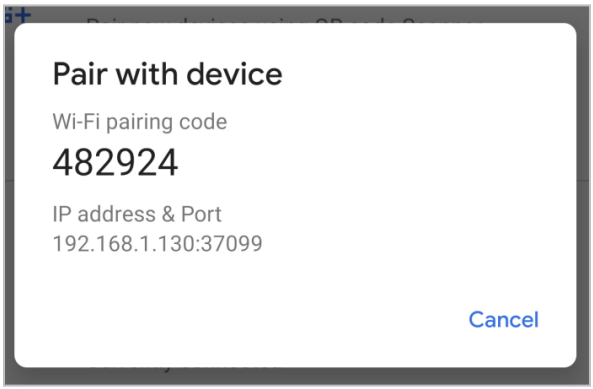
D 性能和质量

无线调试

Android 11 支持通过 Android 调试桥 (adb) 从工作站以无线方式部署和调试应用。例如，您可以将可调试的应用部署到多台远程设备，而无需通过 USB 实际连接您的设备，从而避免常见的 USB 连接问题（例如驱动程序安装方面的问题）。

如需使用无线调试，您需要使用配对码将您的设备与工作站配对。您的工作站和设备必须连接到同一无线网络。如需连接到您的设备，请按以下步骤操作：

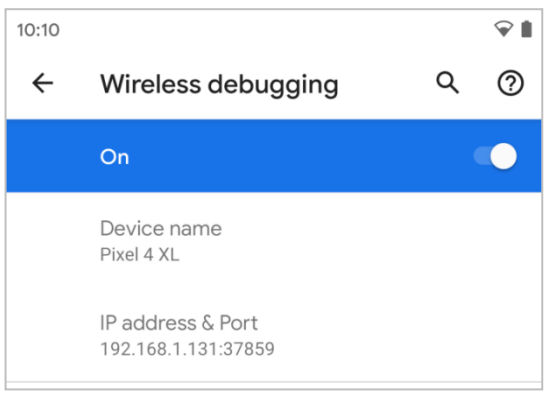
1. 在您的工作站上，更新到最新版本的 SDK 平台工具。
2. 在设备上启用开发者选项。
3. 启用**无线调试**选项。
4. 在询问**要在此网络上允许无线调试吗？**的对话框中，点击**允许**。
5. 选择**使用配对码配对设备**。记下设备上显示的配对码、IP 地址和端口号（参见图片）。
6. 在工作站上，打开一个终端并导航到 android_sdk/platform-tools。
7. 运行 adb pair ipaddr:port。使用第 5 步中的 IP 地址和端口号。



8. 当系统提示时，输入您在第 5 步中获得的配对码。系统会显示一条消息，表明您的设备已成功配对。

```
Enter pairing code: 482924
Successfully paired to 192.168.1.130:37099 [guid=adb-235XY]
```

9. （仅适用于 Linux 或 Microsoft Windows）运行 adb connect *ipaddr:port*。使用**无线调试**下的 IP 地址和端口



（参见下图）。

ADB 增量 APK 安装

在设备上安装大型（2GB 以上）APK 可能需要很长的时间，即使应用只是稍作更改也是如此。ADB (Android 调试桥) 增量 APK 安装可以安装足够的 APK 以启动应用，同时在后台流式传输剩余数据，从而加速这一过程。如果设备支持该功能，并且您安装了最新的 SDK 平台工具，`adb install` 将自动使用此功能。如果不支持，系统会自动使用默认安装方法。

运行以下 adb 命令以使用该功能。如果设备不支持增量安装，该命令将会失败并输出详细的解释。

```
adb install --incremental
```

在运行 ADB 增量 APK 安装之前，您必须先为 APK 签名并创建一个 APK 签名方案 v4 文件。必须将 v4 签名文件放在 APK 旁边，才能使此功能正常运行。

使用原生内存分配器进行错误检测

GWP-ASan 是一种原生内存分配器功能，可帮助查找释放后使用和堆缓冲区溢出错误。您可以全局启用此功能，也可以为应用的特定子进程启用此功能。如需了解详情，请参阅 GWP-Asan 指南。

Neural Networks API 1.3

Android 11 扩展并改进了 Neural Networks API (NNAPI)。

新运算方式

NNAPI 1.3 引入了新的运算数类型 TENSOR_QUANT8_ASYMM_SIGNED，以支持 TensorFlow Lite 的新量化方案。

此外，NNAPI 1.3 还引入了以下新运算：

- QUANTIZED_LSTM
- IF
- WHILE
- ELU
- HARD_SWISH
- FILL
- RANK

新的机器学习控件

NNAPI 1.3 引入了新控件以帮助机器学习流畅运行：

- QoS API：新的 Quality of Service API 使用以下新函数，支持在 NNAPI 中进行优先排序和任务截止时间设定：
    - ANeuralNetworksDevice_wait()
    - ANeuralNetworksCompilation_setPriority()
    - ANeuralNetworksCompilation_setTimeout()

- ANeuralNetworksExecution_setTimeout()

- **内存域输入/输出**：NNAPI 1.3 支持将内存域作为执行的输入和输出。这样可以移除不同系统组件之间不必要的相同数据，从而提高 Android 神经网络的运行时性能。此功能添加了一组用于与 ANeuralNetworksMemoryDesc 和 ANeuralNetworkMemory 对象结合使用的新 NDK API，包括以下函数：

  - ANeuralNetworksMemoryDesc_create()
  - ANeuralNetworksMemoryDesc_free()
  - ANeuralNetworksMemoryDesc_addInputRole()
  - ANeuralNetworksMemoryDesc_addOutputRole()
  - ANeuralNetworksMemoryDesc_setDimensions()
  - ANeuralNetworksMemoryDesc_finish()
  - ANeuralNetworksMemory_createFromDesc()
  - ANeuralNetworksMemory_copy()

  如需了解详情，请参阅神经网络内存域示例。

- **Dependency API 和同步栅栏支持**：NNAPI 1.3 支持使用依赖项实现异步计算，这样可以大大减少调用小型链式模型时产生的开销。此功能添加了以下新函数：

  - ANeuralNetworksEvent_createFromSyncFenceFd()
  - ANeuralNetworksEvent_getSyncFenceFd()

- ANeuralNetworksExecution_startComputeWithDependencies()

- **控制流**：NNAPI 1.3 支持使用新的图表运算 ANEURALNETWORKS_IF 和 ANEURALNETWORKS_WHILE 实现常规控制流，这些运算使用新的 ANEURALNETWORKS_MODEL 运算数类型接受其他模型作为参数。此外，此功能还添加了以下新函数：
    - ANeuralNetworksModel_setOperandValueFromModel()
    - ANeuralNetworks_getDefaultLoopTimeout()
    - ANeuralNetworks_getMaximumLoopTimeout()
    - ANeuralNetworksExecution_setLoopTimeout()

NDK Thermal API

当设备过热时，它们可能会限制 CPU 和/或 GPU，而这可能会以意想不到的方式影响应用。如果应用或游戏包含复杂图形，大量计算或持续网络活动，它们就更容易遇到问题。

在 Android 11 中使用 NDK Thermal API 监控设备上的温度变化，然后采取相应措施以降低耗电量和设备温度。该 API 类似于 Java Thermal API；您可以使用它接收任何热状态更改的通知或直接轮询当前状态。

文本和输入

改进了 IME 转换

Android 11 引入了新的 API 以改进输入法 (IME) 的转换，例如屏幕键盘。这些 API 可让您更轻松地调整应用内容，与 IME 的出现和消失以及状态和导航栏等其他元素保持同步。

如需在聚焦至任何 EditText 时显示 IME，请调用 view.getInsetsController().show(Type.ime()) （您可以在与聚焦的 EditText 相同层次结构中的任何视图上调用此方法，无需专门在 EditText 上调用它）。如需隐藏 IME，请调用 view.getInsetsController().hide(Type.ime())。您可以通过调用 view.getRootWindowInsets().isVisible(Type.ime()) 检查 IME 当前是否可见。

如需同步应用的视图与 IME 的显示和消失，请通过提供 WindowInsetsAnimation.Callback 到 View.setWindowInsetsAnimationCallback() 在视图上设置监听器（您可以在任何视图上设置该监听器，它不一定必须为 EditText）。IME 会调用监听器的 onPrepare() 方法，之后会在转换开始时调用 onStart()。然后，它会在每次转换的过程中调用 onProgress()。转换完成后，IME 会调用 onEnd()。在转换过程中，您随时可以调用 WindowInsetsAnimation.getFraction() 以了解转换的进度。

有关如何使用这些 API 的示例，请参阅新的 WindowInsetsAnimation 代码示例。

控制 IME 动画

您还可以控制 IME 动画或其他系统栏（如导航栏）的动画。如需执行此操作，请先调用 setOnApplyWindowInsetsListener()，为窗口边衬区更改设置新的监听器：

KOTLIN

```
rootView.setOnApplyWindowInsetsListener { rootView, windowInsets ->
    val barsIme = windowInsets.getInsets(Type.systemBars() or
Type.ime())
    rootView.setPadding(barsIme.left, barsIme.top, barsIme.right,
                        barsIme.bottom)

    // We return the new WindowInsets.CONSUMED to stop the insets
being
    // dispatched any further into the view hierarchy. This replaces the
    // deprecated WindowInsets.consumeSystemWindowInsets() and
related
    // functions.
    WindowInsets.CONSUMED
}
```

JAVA

```
mRoot.setOnApplyWindowInsetsListener(new
View.OnApplyWindowInsetsListener() {
  @Override
  public WindowInsets onApplyWindowInsets(View v, WindowInsets
insets) {
```

```
        Insets barsIME = insets.getInsets(Type.systemBars() | Type.ime());
        mRootView.setPadding(barsIme.left, barsIme.top, barsIme.right,
                             barsIme.bottom);

        // We return the new WindowInsets.CONSUMED to stop the insets
being
        // dispatched any further into the view hierarchy. This replaces the
        // deprecated WindowInsets.consumeSystemWindowInsets() and
related
        // functions.
        return WindowInsets.CONSUMED;
    }
});
```

如需移动 IME 或其他系统栏，请调用控制器的 `controlWindowInsetsAnimation()` 方法：

KOTLIN

```
view.windowInsetsController.controlWindowInsetsAnimation(
        Type.ime(),
        1000,
        LinearInterpolator(),
        cancellationSignal,
        object : WindowInsetsAnimationControlListener() {
            fun onReady(controller: WindowInsetsAnimationController,
                        types: Int) {
```

```
                // update IME inset
                controller.setInsetsAndAlpha(Insets.of(0, 0, 0, inset),
                        1f /* alpha */, 0.1 /* fraction progress */)
            }
        }
);
```

JAVA

```
mRoot.getWindowInsetsController().controlWindowInsetsAnimation(
        Type.ime(), 1000, new LinearInterpolator(), cancellationSignal,
        new WindowInsetsAnimationControlListener() {
            @Override
            public void onReady(
                    @NonNull WindowInsetsAnimationController controller,
                    int types
                    ) {
                    // update IME inset
                    controller.setInsetsAndAlpha(Insets.of(0, 0, 0, inset),
                            1f /* alpha */, 0.1 /* fraction progress */);
            }

            @Override
            public void onCancelled() {}
        });
```

ICU 库更新

Android 11 更新了 android.icu 软件包，以使用 ICU 库版本 66，而 Android 10 中使用的是版本 63。新版库包含更新的 CLDR 语言区域数据以及众多对于 Android 中的国际化支持的增强功能。

新版库包含以下主要变更：

- 许多格式化 API 现在都支持可扩展 FormattedValue 的新返回对象类型。
- LocaleMatcher API 在以下方面得到增强：提供了构建器类，支持 java.util.Locale 类型，并且结果类可提供有关匹配的额外数据。
- 现在支持 Unicode 13。

媒体

分配 MediaCodec 缓冲区

Android 11 包含一个新的 MediaCodec API，可让应用在分配输入和输出缓冲区时获得更多控制。这样可以让您的应用更高效地管理内存。

新类：

- MediaCodec.LinearBlock
- MediaCodec.OutputFrame
- MediaCodec.QueueRequest

新方法：

- MediaCodec.getQueueRequest()
- MediaCodec.getOutputFrame()
- MediaCodec.LinearBlock.isCodecCopyFreeCompatible()

此外，MediaCodec.Callback() 中两种方法的行为也发生了变化：

onInputBufferAvailable()

> 如果配置为使用 Block Model API，应用应通过索引使用 MediaCodec.getQueueRequest，并将 LinearBlock/HardwareBuffer 附加到插槽，而不是通过索引调用 MediaCodec.getInputBuffer() 和 MediaCodec.queueInputBuffer()。

onOutputBufferAvailable()

> 应用可以通过索引使用 MediaCodec.getOutputFrame() 获取包含更多信息的 OutputFrame 对象和 LinearBlock/HardwareBuffer 缓冲区，而不是通过索引调用 MediaCodec.getOutputBuffer()。

MediaCodec 低延时解码

Android 11 增强了 MediaCodec，针对游戏和其他实时应用支持低延时解码。您可以将 FEATURE_LowLatency 传递到 MediaCodecInfo.CodecCapabilities.isFeatureSupported()，检查编解码器是否支持低延时解码。

如需启用或停用低延时解码，请执行以下任一操作：

- 使用 MediaCodec.configure() 将新键 KEY_LOW_LATENCY 设置为 0 或 1。
- 使用 MediaCodec.setParameters() 将新参数键 PARAMETER_KEY_LOW_LATENCY 设置为 0 或 1。

> ★ **注意**：支持低延时解码可能需要额外的资源，例如更高的功耗。仅在必要时使用低延时解码。

新的 AAudio 函数 AAudioStream_release()

函数 AAudioStream_close() 会同时释放和关闭音频流。这可能很危险。如果其他进程在音频流关闭后尝试对其进行访问，该进程将会崩溃。

新函数 AAudioStream_release() 会释放音频流，但不会将其关闭。这样会释放其资源并使音频流处于已知状态。该对象将一直存在，直到您调用 AAudioStream_close()。

MediaParser API

MediaParser 是用于媒体提取的新型低级别 API。它比 MediaExtractor 更灵活，并提供对媒体提取功能的额外控制。

通过 USB 设备捕获音频

当没有 RECORD_AUDIO 权限的应用使用 UsbManager 请求直接访问具备音频捕获功能的 USB 音频设备（如 USB 耳机）时，系统会显示一条新的警告消息，要求用户确认设备使用权限。系统会

忽略任何“始终使用”选项，因此应用每次请求访问时，用户都必须确认警告消息并授予相应权限。

为了避免这种行为，您的应用应请求 RECORD_AUDIO 权限。

> ★ **注意**：此行为仅适用于使用 UsbManager API 直接连接到 USB 外围设备的应用。绝大多数媒体播放器、游戏和通信应用使用的是音频 API，因此不会受到此变更的影响。

并发访问麦克风

Android 11 向 AudioRecord、MediaRecorder 和 AAudioStream API 添加了一些新方法。不管选择的用例是什么，这些方法均可启用和停用并发捕获的功能。请参阅共享音频输入。

输出切换器

Android 11 针对使用 Cast 和 MediaRouter API 的应用实现了新行为。

除了可从应用内访问投射选项外，切换选项也显示于系统媒体播放器中。当用户改变视听环境时（例如在厨房中观看视频与在手机上观看之间切换，或者在家中收听音频与在车中收听之间切换），这有助于为用户提供无缝切换设备的流畅体验。请参阅输出切换器。

网络连接

Wi-Fi Passpoint 增强功能

通过 Passpoint，应用可以自动静默地执行身份验证并连接到安全的 Wi-Fi 热点。以 API 级别 30 及更高级别为目标平台的应用可以使用 Passpoint 的以下其他功能。

失效日期强制执行和通知

对个人资料强制执行失效日期可让框架避免使用过期凭据自动连接到接入点，该操作必定会失败。这样可以阻止无线连接，并节省电量和后端带宽。当用户的个人资料位于范围内但已过期时，该功能会向用户显示通知。

FQDN 匹配

允许使用 PerProviderSubscription (PPS) 管理对象 (MO) 中的 Extension/Android 节点，配置独立于接入网络查询协议 (ANQP) 完全限定域名 (FQDN) 的命名 AAA 域。

自签名的私人 CA

对于 Passpoint R1 个人资料，Android 接受采用私人自签名 CA 进行连接身份验证。

允许使用具有相同 FQDN 的多个个人资料

允许安装具有相同 FQDN 的多个 Passpoint 个人资料。FQDN 不用作个人资料的键。需要 FQDN 的现有 Passpoint API（如 remove）会将请求应用于具有相同 FQDN 的所有匹配的个人资料。

允许安装没有根 CA 证书的个人资料

允许使用没有根 CA 证书的个人资料。在这种情况下，系统会根据安装在信任库中的公共根 CA 证书验证 AAA 服务器证书。

改进了家庭网络服务提供商和漫游服务提供商的匹配

系统会匹配家庭网络或漫游网络，而不考虑所通告的身份验证方法。此外，还增加了对 OtherHomePartners 和 HomeOIList 列表的家庭网络匹配功能的支持。

Wi-Fi Suggestion API 扩展

Android 11 扩展了 Wi-Fi Suggestion API，以提高应用的网络管理能力，包括：

- 连接管理应用可以通过允许断开连接请求管理自己的网络。
- Passpoint 网络集成到 Suggestion API 中，可以推荐给用户。
- 通过 Analytics API，您可以获取有关网络质量的信息。

CallScreeningService 更新

从 Android 11 开始，CallScreeningService 可以针对来电请求有关 STIR/SHAKEN 验证状态 (verstat) 的信息。此信息将包含在来电的通话详情中。

如果 CallScreeningService 持有 READ_CONTACTS 权限，当用户通讯录中的号码有来电或向用户通讯录中的号码去电时，应用会收到通知。

Open Mobile API 更新

从 Android 11 开始，Open Mobile API (OMAPI) 有了额外的功能：

- 解析运营商权限的规则。
- 使用以下一项或多项自定义嵌入式安全元件 (eSE) 访问权限或配置 eSE：
    - 系统特许权限
    - 可配置的访问规则应用主数据 (ARA-M) 应用标识符 (AID)
    - 用于重置 OMAPI 读取器的系统 API
- 为读取器提供清晰的指示符，以便应用过滤设备功能。

高性能 VPN

以 API 级别 30 及更高级别为目标平台的应用或在搭载 API 级别 29 及更高级别的设备上运行的应用可以将 IKEv2/IPsec 应用于 VPN（包括用户配置的 VPN 和基于应用的 VPN）。

VPN 本身在操作系统上运行，从而简化了在应用中建立 IKEv2/IPset VPN 连接所需的代码。

每个进程的网络访问控制

如需了解如何针对各进程启用网络访问权限，请参阅管理网络使用情况。

允许安装的多种 Passpoint 配置具有相同的 FQDN

从 Android 11 开始，您可以使用 PasspointConfiguration.getUniqueId() 获取 PasspointConfiguration 对象的专有标识符，这样可让使用应用的用户安装多个具有相同完全限定域名（FQDN）的配置文件。

当运营商在其网络上部署多个移动设备国家/地区代码（MCC）和移动网络代码（MNC）组合，但只有一个 FQDN 时，此功能非常有用。在 Android 11 及更高版本中，当用户安装具有 MCC 或 MNC 的 SIM 卡时，可以安装多个具有相同 FQDN（它将与家庭网络服务提供商提供的网络匹配）的配置文件。

★ 注意：各配置由唯一键进行标识，该键取决于配置的内容。如需更新现有配置文件，您必须使用 WifiManager.removePasspointConfiguration() 将其移除。如果不移除现有配置，会导致添加一个包含两种配置的新配置文件。

GNSS 天线支持

Android 11 引入了 GnssAntennaInfo 类，让您的应用能够更多地利用全球导航卫星系统（GNSS）提供的厘米精度定位。

如需了解详情，请参阅有关天线校准信息的指南。

图形

NDK 图像解码器

NDK ImageDecoder API 提供了一种标准 API，供 Android C/C++ 应用直接解码图像。应用开发者不再需要使用框架 API（通过 JNI）或捆绑第三方图像解码库。有关详情，请参阅图像解码器开发者指南。

Frame rate API

Android 11 提供了一个 API，可让应用告知系统其预期帧速率，从而减少支持多个刷新率的设备上的抖动。有关如何使用此 API 的信息，请参阅帧速率指南。

请求并检查低延时支持

特定的显示屏可以执行图形后期处理，例如某些外部显示屏和电视。此类后期处理改善了图形质量，但可能会增加延时。支持 HDMI 2.1 的新款显示屏具有自动低延时模式（ALLM，也称为游戏模式），该模式可以通过关闭后期处理以最大限度地缩短延时。如需详细了解 ALLM，请参阅 HDMI 2.1 规范。

窗口可以请求使用自动低延时模式（如果可用）。ALLM 对于游戏和视频会议等应用特别有用，因为对于这些应用而言，低延时的重要性要高于拥有最佳的图形质量。

如需开启或关闭最低限度的后期处理，请调用 Window.setPreferMinimalPostProcessing()，或将窗口的 preferMinimalPostProcessing 属性设置为 true。并非所有的显示屏都支持最低限度的

后期处理；如需了解某个显示屏是否支持该功能，可调用新方法 Display.isMinimalPostProcessingSupported()。

> ★ **注意**：如果用户停用最低限度的后期处理，或者显示屏不支持低延时模式，那么调用 Window.setPreferMinimalPostProcessing() 不会有任何作用。

高性能图形调试层注入

应用现在可以将外部图形层（GLES、Vulkan）加载到原生应用代码中，可以在不产生性能开销的前提下，提供与可调试应用相同的功能。在使用 GAPID 等工具对应用进行性能剖析时，此功能尤为重要。如需对应用进行性能剖析，只需要在应用清单文件中添加以下元数据元素，而无需让应用变成可调试应用：

```
  <application ... >
    <meta-data
android:name="com.android.graphics.injectLayers.enable"
android:value="true" />
</application>
```

图片和相机

在主动拍摄期间关闭通知提示音和振动

从 Android 11 开始，在主动使用相机时，您的应用可以使用 setCameraAudioRestriction() 以仅关闭振动、同时关闭声音和振动或都不关闭。

Android 模拟器中的相机支持扩展

如需了解自 Android 11 起模拟器中的相机支持扩展，请参阅相机支持。

支持并发使用多个摄像头

Android 11 添加了 API 以查询对同时使用多个摄像头（包括前置摄像头和后置摄像头）的支持。

如需在运行应用的设备上检查支持情况，请使用以下方法：

- getConcurrentCameraIds() 可返回摄像头 ID 组合 Set，这些组合可与有保证的数据流组合并发进行流式传输（如果它们是由同一应用进程配置的）。
- isConcurrentSessionConfigurationSupported() 可查询摄像头设备是否可以并发支持相应的会话配置。

更好地支持包含多个帧的 HEIF 图片

从 Android 11 开始，如果您调用 ImageDecoder.decodeDrawable() 并传递包含帧序列的 HEIF 图片（如动画或连拍照片），则该方法会返回包含整个图片序列的 AnimatedImageDrawable。在较低版本的 Android 系统中，该方法会返回仅包含单个帧的 BitmapDrawable。

如果 HEIF 图片包含的多个帧不在一个序列中，您可以通过调用 MediaMetadataRetriever.getImageAtIndex() 检索各个帧。

无障碍功能

面向无障碍服务开发者的更新

如果您创建自定义无障碍服务，可以在 Android 11 中使用以下功能：

- 在无障碍服务的面向用户的解释中，除了纯文本之外，现在还允许使用 HTML 和图片。这种灵活性可让您更轻松地向最终用户解释您的服务有何功能以及对他们有何帮助。
- 如需使用比 contentDescription 在语义上更有意义的界面元素的状态说明，请调用 getStateDescription() 方法。
- 如需请求触摸事件绕过系统的触摸浏览器，请调用 setTouchExplorationPassthroughRegion()。同样，如需请求手势绕过系统的手势检测器，请调用 setGestureDetectionPassthroughRegion()。
- 您可以请求 IME 操作（如“输入”和“下一个”），以及不启用 FLAG_SECURE 标记的窗口的屏幕截图。

E 其他功能

应用进程退出原因

Android 11 引入了 ActivityManager.getHistoricalProcessExitReasons() 方法，用于报告近期任何进程终止的原因。应用可以使用此方法收集崩溃诊断信息，例如进程终止是由于 ANR、内存问题还是其他原因所致。此外，您还可以使用新的 setProcessStateSummary() 方法存储自定义状态信息，以便日后进行分析。

getHistoricalProcessExitReasons() 方法会返回 ApplicationExitInfo 类的实例，该类包含与应用进程终止相关的信息。通过对此类的实例调用 getReason()，您可以确定应用进程终止的原因。例如，返回值为 REASON_CRASH 表示您的应用中发生了未处理的异常。如果您的应用需要确保退出事件的唯一性，它可以维护一个应用专用的标识符，如基于来自 getTimestamp() 方法的时间戳的哈希值。

★ **注意**：某些设备无法报告内存不足终止事件。在这些设备上，getHistoricalProcessExitReasons() 方法会返回 REASON_SIGNALED 而不是 REASON_LOW_MEMORY，并

且 getStatus() 的返回值为 SIGKILL。

如需检查设备是否可以报告内存不足终止事件，请调用 ActivityManager.isLowMemoryKillReportSupported()。

资源加载器

Android 11 引入了一个新 API，允许应用动态扩展资源的搜索和加载方式。新的 API 类 ResourcesLoader 和 ResourcesProvider 主要负责提供新功能。两者协同作用，可以提供额外的资源，或修改现有资源的值。

ResourcesLoader 对象是向应用的 Resources 实例提供 ResourcesProvider 对象的容器，而 ResourcesProvider 对象提供从 APK 和资源表加载资源数据的方法。

此 API 的一个主要用例是自定义资源加载。您可以使用 loadFromDirectory() 创建一个 ResourcesProvider，用于重定向基于文件的资源的解析，从而让其搜索特定目录，而不是应用 APK。您可以通过 AssetManager API 类中的 open() 系列方法访问这些资源，就像访问 APK 中绑定的资源一样。

APK 签名方案 v4

Android 11 添加了对 APK 签名方案 v4 的支持。此方案会在单独的文件 (apk-name.apk.idsig) 中生成一种新的签名，但在其他方

面与 v2 和 v3 类似。没有对 APK 进行任何更改。此方案支持 ADB 增量 APK 安装，这样会加快 APK 安装速度。

动态 intent 过滤器

如需接收 intent，应用必须通过在其清单中定义 intent 过滤器，在编译时声明它能够接收哪些类型的数据。在 Android 10 及更低版本中，应用无法在运行时更改其 intent 过滤器。这对于虚拟化应用（如虚拟机和远程桌面）而言是一个问题，因为这些应用无法确切得知用户将在它们内部安装什么软件。

Android 11 引入了 MIME 组，这是一个新的清单元素，可让应用在 intent 过滤器中声明一组动态的 MIME 类型，并在运行时以编程方式对其进行修改。如需使用 MIME 组，请使用新的 android:mimeGroup 属性在应用清单中添加一个数据元素：

```
<intent-filter>
  <action android:name="android.intent.action.SEND"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <data android:mimeGroup="myMimeGroup"/>
</intent-filter>
```

android:mimeGroup 属性的值是任意字符串 ID，用于在运行时标识 MIME 组。您可以通过将某个 MIME 组的 ID 传递给 PackageManager API 类中的以下新方法，访问和更新该 MIME 组的内容：

- getMimeGroup()
- setMimeGroup()

如果您以编程方式将 MIME 类型添加到 MIME 组，其运作方式与清单中明确声明的静态 MIME 类型完全相同。

注意：mimeGroup 字符串是基于每个软件包定义的。在同一软件包中，您可以在多个 intent 过滤器或组件中使用相同的 mimeGroup 字符串以声明它们之间共享的 MIME 组。不同的软件包不能共享 MIME 组，但它们可以使用相同的 mimeGroup 字符串，而不相互干扰。

自动填充增强功能

Android 11 改进了自动填充服务。

AssistStructure.ViewNode 中的提示标识符

对自动填充服务来说，根据视图的属性计算视图的签名哈希值通常很有用。在计算签名哈希值时，视图提示是一个非常值得参考的属性，但提示字符串可能会随着手机的语言区域而发生变化。为了解决此问题，Android 11 使用新的 `getHintIdEntry()` 方法扩展了 `AssistStructure.ViewNode`，该方法会返回视图提示文本的资源标识符。此方法提供一个与语言区域无关的值，可用于计算签名哈希值。

提供了数据集的事件

为了帮助自动填充服务提高建议内容的质量，Android 11 提供了一种方法以识别自动填充服务提供了数据集但用户未选择任何数据集的情况。在 Android 11 中，`FillEventHistory` 会报告一种新的 `TYPE_DATASETS_SHOWN` 事件类型。每当自动填充服务向用户提供一个或多个数据集时，`FillEventHistory` 就会记录此类型的事

件。自动填充服务可以将这些事件与现有的 TYPE_DATASET_SELECTED 事件结合使用来确定用户是否选择了任何提供的自动填充选项。

IME 集成

键盘及其他 IME 现在可以在建议栏或类似的界面中以内嵌方式显示自动填充建议，而不是在下拉菜单中显示这些建议。为了保护密码和信用卡号等敏感信息，系统会将建议显示给用户，但在用户选择某条建议之前，IME 并不知道这些建议。如需了解 IME 和密码管理器如何支持此功能，请参阅将自动填充功能与键盘集成。

与内容捕获服务共享数据

从 Android 11 开始，应用可以与设备的内容捕获服务共享数据。借助此功能，设备可以更轻松地提供情境智能，例如显示用户环境中正在播放的歌曲的名称。

如需将应用中的数据共享给内容捕获服务，请对 ContentCaptureManager 的实例调用 shareData() 方法。如果系统接受数据共享请求，应用会收到将与内容捕获服务共享的只写文件描述符。

1.5.2 向您的应用添加 5G 功能

Android 11 添加了在您的应用中支持 5G 的功能。本主题介绍了该功能，并简要说明了向您的应用添加 5G 专用功能如何改善用户体验。

A 针对 5G 构建

在决定如何与 5G 互动时，思考一下您试图打造什么样的体验。5G 可通过一些方法增强您的应用，其中包括：

- 由于 5G 在速度和延迟方面的改进，自动使当前的体验更快更好。
- 提升用户体验，如通过显示 4k 视频或下载分辨率更高的游戏资产。
- 在确认增加的流量消耗不会让用户付费后，添加通常仅通过 WLAN 提供的体验，如主动下载一般为不按流量计费的 WLAN 保留的内容。
- 提供 5G 独有的体验，这种体验只能在高速度且低延迟的网络上实现。

B 5G 功能

Android 11 引入了以下功能变更和增强功能：

- 按流量计费性
- 5G 检测
- 带宽估测

检查按流量计费性

NET_CAPABILITY_TEMPORARILY_NOT_METERED 是 Android 11 中添加的一项功能，可根据移动网络运营商提供的信息，告知您正在使用的网络是否不按流量计费。

该新标记与 NET_CAPABILITY_NOT_METERED 一起使用。该现有标记指示网络是否始终不按流量计费，并且同时适用于 WLAN 和移动网络连接。

这两个标记之间的区别在于，在网络类型不变的情况下，NET_CAPABILITY_TEMPORARILY_NOT_METERED 可能会发生变化。以 Android 11 为目标平台的应用可以使用 NET_CAPABILITY_TEMPORARILY_NOT_METERED 标记。在搭载 Android 9 及更低版本的设备上，操作系统不会报告该标记。对于在 Android 10 上运行的应用，此标记可能可用，具体取决于运行应用的设备。

一旦确定当前网络暂时或永久不按流量计费，您便可以显示分辨率更高的内容（如 4k 视频）、上传日志、备份文件，以及主动下载内容。

下面几部分介绍了向您的应用添加按流量计费性检查的步骤。

注册网络回调

使用 ConnectivityManager.registerDefaultNetworkCallback() 注册一个网络回调，以监听 NetworkCapabilities 何时发生更改。您可以通过替换 NetworkCallback 中的 onCapabilitiesChanged() 方法来检测 NetworkCapabilities 的更改。

registerDefaultNetworkCallback() 会使注册的回调在注册后立即触发，从而为应用提供有关当前状态的信息。将来的回调对于应用在状态从不按流量计费更改为按流量计费或者从按流量计费更改为不按流量计费时采取适当的措施至关重要。

检查按流量计费性

使用在网络回调中收到的 NetworkCapabilites 对象来检查以下代码的输出：

KOTLIN

```
NetworkCapabilities.hasCapability(NET_CAPABILITY_NOT_METERED) ||
    NetworkCapabilities.hasCapability(NET_CAPABILITY_TEMPORARILY_
    NOT_METERED)
```

JAVA

```
NetworkCapabilities.hasCapability(NET_CAPABILITY_NOT_METERED) ||
    NetworkCapabilities.hasCapability(NET_CAPABILITY_TEMPORARILY_
    NOT_METERED)
```

如果值为 true，则您可以将网络视为不按流量计费。

其他注意事项

使用此功能时，请注意以下几点：

- 使用 NET_CAPABILITY_TEMPORARILY_NOT_METERED 标记要求您针对 Android 11 SDK 编译您的应用。
- NET_CAPABILITY_NOT_METERED 功能是网络上的永久性功能。如果具有此功能的网络失去此功能（变为按流量计费），该网络会自动断开连接。
- 相比之下，NET_CAPABILITY_TEMPORARILY_NOT_METERED 可以在不断开网络连接的情况下在网络上发生变化。因此，应用必须监听 onCapabilitiesChanged() 回调，以便在网络恢复到其按流量计费状态（失去 NET_CAPABILITY_TEMPORARILY_NOT_METERED 功能）时进行处理。
- 一个网络不能同时具有 NET_CAPABILITY_NOT_METERED 和 NET_CAPABILITY_TEMPORARILY_NOT_METERED。

5G 检测

从 Android 11 开始，您可以使用基于回调的 API 调用来检测设备是否连接到了 5G 网络。您可以检查连接的是 5G NR（独立）网络，还是 NSA（非独立）网络。

> ★ **注意**：虽然您可以检测是否连接到了 5G 网络，但不能根据此信号来假定按流量计费性、连接速度或带宽。

此 API 调用的一些用途可能包括：

- 在您的应用中显示 5G 品牌信息，以强调您提供的是独一无二的 5G 体验。
- 只有连接到 5G 网络时，才能在应用中激活独一无二的 5G 体验。您应将此状态检查与检查按流量计费性搭配使用。
- 为了分析目的而跟踪 5G 连接。

如需在没有 5G 设备的情况下测试 5G 检测，您可以使用添加到 Android SDK 模拟器的功能。

检测 5G

调用 TelephonyManager.listen() 并传入 LISTEN_DISPLAY_INFO_CHANGED，以确定用户是否连接到了 5G 网络。替换 onDisplayInfoChanged() 方法，以确定应用连接到的网络类型：

返回类型	网络
OVERRIDE_NETWORK_TYPE_LTE_ADVANCED_PRO	高级专业版 LTE (5Ge)
OVERRIDE_NETWORK_TYPE_NR_NSA	NR (5G) - 5G Sub-6 网络
OVERRIDE_NETWORK_TYPE_NR_NSA_MMWAVE	5G+/5G UW - 5G mmWave 网络

★ 注意：您的应用必须具有 READ_PHONE_STATE 权限，才能使用此 API。

带宽估测

带宽估测使用您在确定按流量计费性时使用的 NetworkCapabilities 对象。您可以使用该对象获取带宽估测值。

带宽估测方法 getLinkDownstreamBandwidthKbps() 和 getLinkUpstreamBandwidthKbps() 的可靠性和准确性在 Android 11 中得到了改进，这是因为，为了适应 5G 而进行了框架支持的升级和平台/调制解调器问题修复。

带宽默认值仅提供关于应用启动的指导。这应该可以帮助您处理“空闲时启动”的情况。您的应用应衡量用户开始与其互动后的性能，并动态地调整其流式传输行为。例如，您可以根据启动时的带宽估测来选择要提供的视频分辨率。随着用户使用应用，继续检查估测值；随着其连接类型和强度的变化，相应地调整应用的行为。

★ **注意**：仅凭带宽估测值无法判断用户是否连接到了 5G 网络。如需确定这一点，请参阅 5G 检测。

1.5.3 强制门户 API 支持

从 Android 11 Beta 版 2 开始，系统支持 RFC7710bis 中所述的一部分功能以及关联的强制门户 API。

该 API 为接入点提供了一种可靠的方法来将其自身标识为强制门户。此外，它还支持接入点向用户发布信息（如会话和信息中心信息）的新用例。

A 改进了强制门户检测

从 Android 5.0（API 级别 21）开始，Android 设备就已能够检测强制门户，并通知用户他们需要登录网络才能访问互联网。之前是通过对已知目标网站（如 connectivitycheck.gstatic.com）进行明文 HTTP 探测来检测强制门户，如果探测收到 HTTP 重定向，设备就会假定相应的网络是强制门户。这种方法可能不可靠，因为没有标准的网址可供探测，并且强制门户网络可能会错误地允许或阻止（而不是重定向）此类探测。该 API 允许门户提供一个表明要求登录的正信号，以及要登录的网址。

Android 11 支持 DHCP 选项 114，如 RFC7710bis 中所述。我们可能会在未来的更新中添加对路由器通告选项的支持。如果设备在 DHCP 握手期间通过该选项获得强制门户 API 网址，设备会在连接后立即提取 API 内容，按照强制门户 API 要求，如果网络是强制门户，还会提示用户登录。

如果该 API 不可用或未通告任何门户，系统会像以前一样，继续使用 HTTP/HTTPS 探测来检测门户并验证互联网连接。

B 信息中心发布的信息

Android 11 支持强制门户 API 中定义的 venue-info-url。此网址允许用户在其浏览器中获取有关接入点信息中心的特定于上下文的信息。默认情况下，如果用户选择打开此网址，可以在登录后从通知中打开，或从网络设置中打开。

图 1. 如果网络提供了信息中心网址，系统会弹出一条通知，允许用户访问该网页

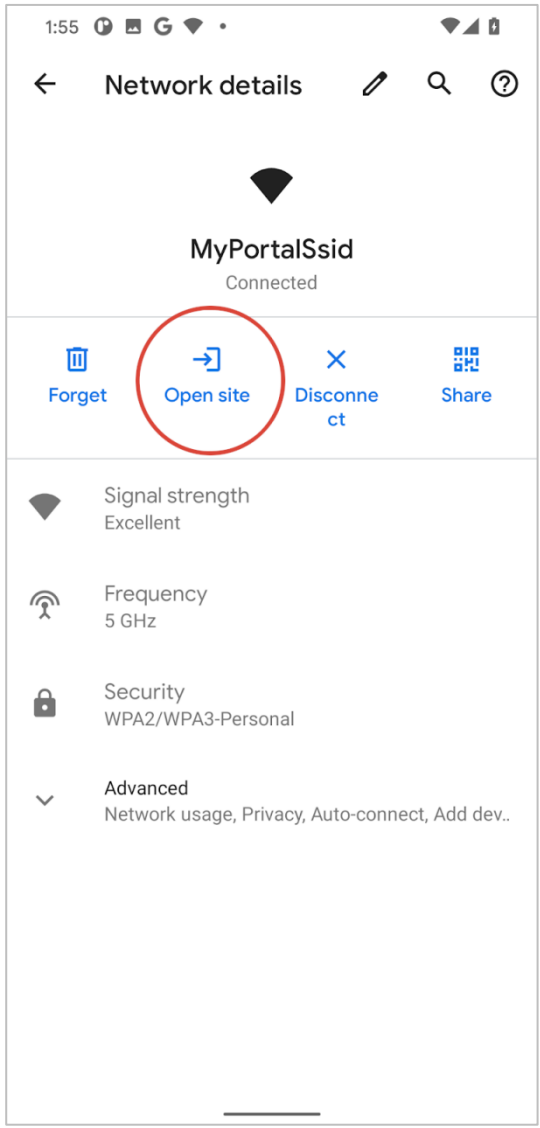


图 2. 用于从“网络详情”屏幕中打开网站的按钮

C

未来的用例

虽然 Android 11 在发布时仅支持强制门户 API 中的一组基本功能，但在发布后可能会通过 Google Play 系统更新将新功能分发给设备。我们建议网络运营商实现强制门户 API，同时注意未来可能的改进：

- 会话时间（seconds-remaining）当前在默认的**设置**应用中用来告知用户在门户上还剩多长时间。通过登录网址延长会话的能力（can-extend-session）也可以通过该 API 表示，以允许系统通知用户已到期或即将到期的会话。
- 数据上限（bytes-remaining）可以通过该 API 通告，以允许用户跟踪剩余数据。

1.5.4 安全共享大型数据集

从 Android 11（API 级别 30）开始，系统会缓存可供多个应用访问的大型数据集，为机器学习和媒体播放等用例提供支持。此功能有助于减少网络中和磁盘上的数据冗余。

当您的应用需要访问大型共享数据集时，可以先查找是否有这类缓存的数据集（称为共享数据 blob），然后再决定是否下载新副本。应用可以通过 BlobStoreManager 中的 API 访问此共享数据集功能。

系统维护着共享数据 blob 并控制着哪些应用可以访问它们。当您的应用提供数据 blob 时，您可以通过调用以下方法之一指明应该对其具有访问权限的其他应用：

- 如需向设备上的一组特定应用授予访问权限，请将这些应用的软件包名称传递到 allowPackageAccess() 中。
- 要仅允许证书签名所用密钥与您的应用所用密钥相同的应用（例如您管理的应用套件），请调用 allowSameSignatureAccess()。
- 如需向设备上的所有应用授予访问权限，请调用 allowPublicAccess()。

A

访问共享数据 blob

系统使用 BlobHandle 对象表示每个共享数据 blob。每个 BlobHandle 实例都包含相应数据集的加密安全哈希值和一些识别性详细信息。

如需访问共享数据 blob，请从服务器下载识别性详细信息。使用这些详细信息检查系统上是否已存在相应数据集。

下一步取决于是否已存在相应数据。

已存在数据集

如果设备上已存在相应数据集，请从系统访问该数据集，如以下代码段所示：

KOTLIN

```
val blobStoreManager =
        getSystemService(Context.BLOB_STORE_SERVICE) as
BlobStoreManager
// The label "Sample photos" is visible to the user.
val blobHandle = BlobHandle.createWithSha256(sha256DigestBytes,
        "Sample photos",
        System.currentTimeMillis() + TimeUnit.DAYS.toMillis(1),
        "photoTrainingDataset")
try {
    val input = ParcelFileDescriptor.AutoCloseInputStream(
            blobStoreManager.openBlob(blobHandle))
```

```
    useDataset(input)
}
```

JAVA

```
BlobStoreManager blobStoreManager =
        ((BlobStoreManager)
getSystemService(Context.BLOB_STORE_SERVICE));
if (blobStoreManager != null) {
    // The label "Sample photos" is visible to the user.
    BlobHandle blobHandle = BlobHandle.createWithSha256(
            sha256DigestBytes,
            "Sample photos",
            System.currentTimeMillis() + TimeUnit.DAYS.toMillis(1),
            "photoTrainingDataset");
    try (InputStream input = new
ParcelFileDescriptor.AutoCloseInputStream(
            blobStoreManager.openBlob(blobHandle))) {
        useDataset(input);
    }
}
```

不存在数据集

如果不存在相应数据集，请从服务器下载该数据集并将其提供给系统，如以下代码段所示：

KOTLIN

```
val sessionId = blobStoreManager.createSession(blobHandle)
try {
    val session = blobStoreManager.openSession(sessionId)
    try {
        // For this example, write 200 MiB at the beginning of the file.
        val output = ParcelFileDescriptor.AutoCloseOutputStream(
                session.openWrite(0, 1024 * 1024 * 200))
        writeDataset(output)

        session.apply {
            allowSameSignatureAccess()
            allowPackageAccess(your-app-package,
                    app-certificate)
            allowPackageAccess(some-other-app-package,
                    app-certificate)
            commit(mainExecutor, callback)
        }
    }
}
```

JAVA

```
long sessionId = blobStoreManager.createSession(blobHandle);
try (BlobStoreManager.Session session =
        blobStoreManager.openSession(sessionId)) {
    // For this example, write 200 MiB at the beginning of the file.
    try (OutputStream output = new
ParcelFileDescriptor.AutoCloseOutputStream(
            session.openWrite(0, 1024 * 1024 * 200)))
        writeDataset(output);
        session.allowSameSignatureAccess();
        session.allowPackageAccess(your-app-package,
                app-certificate);
        session.allowPackageAccess(some-other-app-package,
                app-certificate);
        session.commit(getMainExecutor(), callback);
    }
}
```

1.5.5 联系人与对话

“人与对话”计划是一项多年期 Android 计划，目的是在手机的系统界面中提升人与对话的比重。这种重视基于这样一个事实：对于形形色色的用户而言，与他人的沟通和互动仍然是绝大多数人最看重也是最重要的功能领域。

Android 11 中引入了许多功能来支持“人与对话”计划。

A 对话空间

在许多手机上，通知栏的顶部都有一个单独的部分，其中仅包含与他人的实时对话（例如通话和聊天消息，包括群聊）。此空间内的通知在外观和行为上不同于许多手机上的非会话通知：

- 设计不同，着重强调代表用户的头像，再加上进行对话所用的应用。
- 点按通知即可在应用中打开对话（如果对话此前以对话泡形式显示，则会以对话泡的形式中打开），点按文字插入点即可将通知栏中的新消息展开到完整篇幅，同时显示完整选项列表。
- 提供了对话专用的操作（某些操作通过长按来执行）：
    - 将此对话标记为优先
    - 将此对话提升为对话泡（仅当应用支持对话泡时才会显示）

- 将此对话的通知设为静音
- 为此对话设置自定义提示音或振动

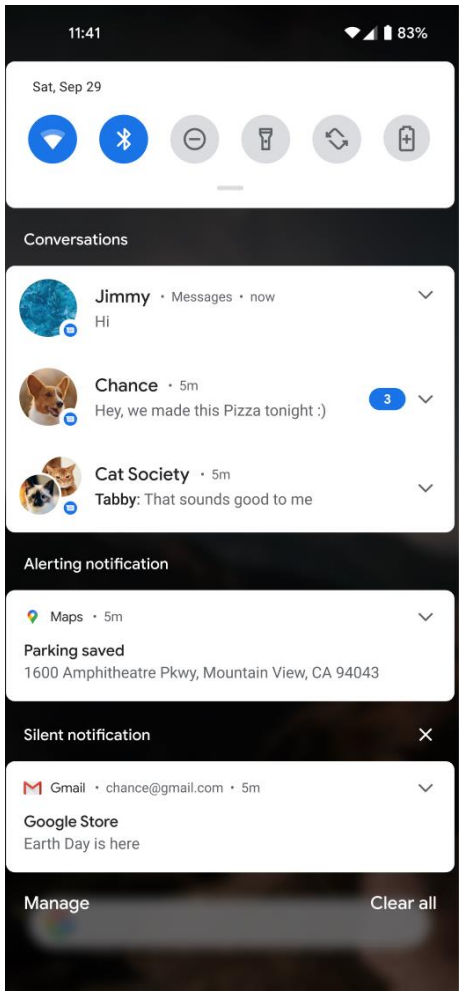


图 1：对话空间。

B

对话泡中的对话

从 Android 11 开始，对话泡可以从“对话”部分中的通知启动。只有带关联快捷方式的通知才能以对话泡形式显示。如果对话在通知栏中标记为“重要”或触发了对话泡显示方式，系统将自动以对话泡形式显示这些对话。

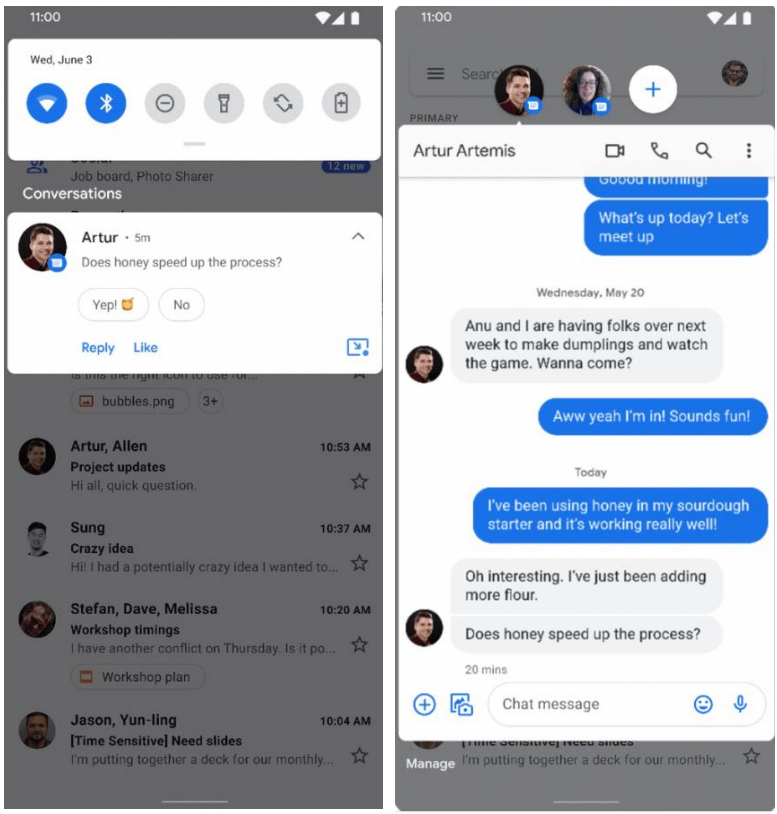


图 2：通知以对话泡形式从抽屉式通知栏中启动。

C 对话快捷方式

对话的快捷方式会出现在启动器中，并且会与系统共享表单中长期存在的共享快捷方式一起出现。

D API 准则

本部分介绍了一些 API，这些 API 旨在支持在您的应用中使用系统提供的人与对话显示空间。

对话的快捷方式

为了参与这项以对话为中心的计划，应用需要向系统提供长期存在的快捷方式。我们强烈建议您使用长期存在的共享快捷方式。如有需要，您也可以在 Android 11 中使用动态快捷方式，但将来，我们可能会移除这个选项。

如需发布对话的快捷方式，请调用现有的 ShortcutManager 方法 setDynamicShortcuts()、addDynamicShortcuts() 或 pushDynamicShortcut()（可自动为开发者管理快捷方式限制）。此快捷方式必须长期存在并附有关于一个人或多个人的 Person 数据，用于标识对话中的其他参与者。我们还建议您为快捷方式设置 LocusId，并使用该 LocusId 注释应用内 Activity 和 Fragment。这样做有助于系统根据应用的使用情况对对话进行准确排名。

如果某段对话已不存在，应用可以使用 removeLongLived Shortcuts() 删除相应快捷方式；这样做会让系统删除与这段对话关联的所有数据。虽然快捷方式可移除，但除非绝对必要，否则应用**不得移除缓存的快捷方式**；缓存某个快捷方式或许是因为用户与之互动来改变体验，移除该快捷方式会撤消这些更改，从而引起用户的不满。

> ★ **注意**：如果移除缓存的快捷方式也会移除用户专用的对话通知渠道，那么设置中删除的类别计数会递增。

对话通知

如果满足以下条件，会将通知视为对话通知：

- 通知使用 MessagingStyle。
- （只有在应用以 Android 11 或更高版本为目标平台时适用）通知与长期存在的有效动态或缓存共享快捷方式关联。通知可以通过调用 setShortcutId() 或 setShortcutInfo() 来设置此关联。如果应用以 Android 10 或更低版本为目标平台，通知就不必与快捷方式关联，如回退选项部分中所述。
- 在发布时，用户没有通过通知渠道设置使对话部分中的对话降位。

借助 LocusId 对应用内对话进行排名

设备端智能技术可确定用户最有可能感兴趣的对话。有许多重要的信号，其中最重要的两个是每段对话中对话会话的新近度和频率。

如果正确地标记了通过启动器快捷方式或在通知中与对话进行的互动，系统就会知道这些互动。不过，系统不知道完全发生在应用中的对话，除非也标记了这些互动。因此，我们强烈建议您将 LocusId 附加到快捷方式，并使用关联的 LocusId 注释应用内 Activity 或 Fragment。使用 LocusId 可让建议系统正确地对对话进行排名。如果您使用 setShortcutInfo() 将对话与快捷方式关联，对话系统会自动附加适当的 LocusId。

E 对话空间对以 Android 10 或更低版本为目标平台的应用有哪些要求

如果应用并非以 Android 11 为目标平台，其消息仍可呈现在对话空间中。不过，应用仍必须满足某些要求。本部分介绍了对这些应用有哪些要求，以及应用不满足这些要求时表现出的回退行为。

> ★ **注意**：如果应用以 Android 11 或更高版本为目标平台，则必须遵循 API 准则部分中所述的要求，才能使其消息出现在对话空间中。

加入消息传递空间的核心要求是应用必须实现 MessagingStyle 通知，并且此类通知必须引用一个**长期存在的快捷方式**，该快捷方式在发布通知时发布。满足这些要求的通知会出现在对话空间中，其行为如下：

- 通知以**对话样式**显示
- 提供了**对话泡**按钮（如果已实现）

- 以内嵌方式提供了对话专用的功能

如果通知不满足这些要求，则平台会使用回退选项来设置通知的格式。如果通知满足任意一种回退情况的要求，则会以特殊格式显示在对话空间中。如果通知不符合任意一个回退选项的条件，则不会显示在对话空间中。

回退：如果使用了 MessagingStyle 但未提供快捷方式

如果应用以 Android 10 或更低版本为目标平台，并且某条通知使用了 MessagingStyle 但未将消息与快捷方式关联，则该通知会显示在对话空间中，其行为如下：

- 通知以对话样式显示
- 没有提供“对话泡”按钮
- 没有以内嵌方式提供对话专用的功能

回退：如果未使用 MessagingStyle 但将应用识别为即时通讯应用

如果某条通知未使用 MessagingStyle，但平台将应用识别为即时通讯应用，并且该通知的 category 参数设为 msg，则该通知会显示在对话空间中，其行为如下：

- 通知以 Android 11 之前版本中的旧样式显示
- 没有提供“对话泡”按钮
- 没有以内嵌方式提供对话专用的功能

F

指导、使用和测试

本部分提供了有关如何使用和测试对话功能的一般指导。

我应该在何时使用对话?

对话通知和相关的快捷方式旨在改善**实时对话**的用户体验。例如，短信、文字聊天和通话就是用户期望能够快速沟通的实时对话。用户对电子邮件以及与对话无关的活动没有这样的期望。

我们为用户提供了相关功能，如果他们觉得给定的对话所在位置不合适，能够从对话部分中将其移除。

提供良好的快捷方式

如果您的应用以 Android 11 或更高版本为目标平台，为了让消息出现在对话空间中，您必须提供快捷方式。应提供 AdaptiveIcon Drawable 作为快捷方式的图标，否则您的快捷方式头像可能会被无意中裁剪；如需了解详情，请参阅提供快捷方式图像。

您的快捷方式会在不同的系统界面中进行排名，包括系统共享表单（如果它是共享快捷方式）。详细了解如何获得最佳排名并帮助系统提升您的快捷方式。

测试对话通知和快捷方式

如果您遵循对话空间准则，对话应自动出现在**对话空间**中。 您可以长按通知来验证快捷方式是否已与通知正确集成。如果正确实现了集成，则界面会显示与对话相关的操作。如果通知未与快捷方式关联，则界面会显示一些文字，说明应用不支持对话功能。

长按应用启动器时，会显示已添加的快捷方式。请务必测试这些快捷方式是否可以使您转到应用中的正确位置。

共享您的共享快捷方式可以接收的内容时，已添加的共享快捷方式会显示在系统共享表单的直接共享行中。

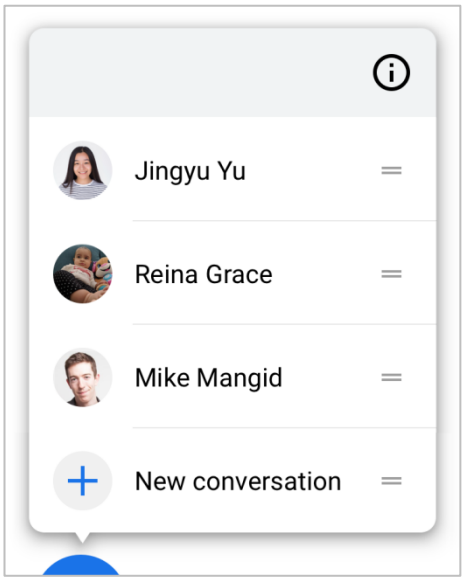


图 3: 您可以长按对话通知，然后检查是否出现了对话菜单，由此验证是否正确配置了对话通知。

1.5.6 在企业中使用 Android

此部分简要介绍了 Android 11 推出的新企业 API、功能和行为变更。如需了解可能会影响您的应用的其他变更，请参阅 Android 11 行为变更页面（针对以 Android 11 为目标平台的应用和所有应用）。

工作资料

Android 11 中提供了可供工作资料使用的以下新功能。

面向公司自有设备的工作资料增强功能

Android 11 改进了对公司自有设备上的工作资料的支持。如果使用 Android 10 中新增的配置工具在设置向导中添加工作资料，设备会被识别为归公司所有，而且还会有更广泛的资产管理和设备安全政策可供设备政策控制器 (DPC) 使用。借助这些功能，既可以更轻松地管理公司自有设备上的工作用途和个人用途，同时又能保持对工作资料的隐私保护。

如果使用任何其他方法将工作资料添加到设备，Android 11 会将设备识别为个人所有。对于个人所有设备上的工作资料，可供使用的行为和功能保持不变。

升级到 Android 11 的设备

在 Android 11 上，完全托管设备中的工作资料将升级，工作资料体验会得到提升。对于客户而言，这意味着设备的隐私权益将获得改善，而且客户能够在个人所有设备和公司自有设备上享受到单一工作资料体验的一致性，而无需重新注册完全托管设备上的旧工作资料。如果您愿意，也可以在升级前移除工作资料，这样就能在整个升级过程中保持完全托管设备体验。

客户可以与 EMM 联系，确保自己的设备已准备好升级到 Android 11。EMM 可在 Android Enterprise EMM 提供商社区（需要登录）中找到更详细的迁移指南。

提升用户体验

Android 9 中为默认启动器引入的单独工作标签页和个人标签页已扩展到更多设备功能。在 Android 11 中，设备制造商可以针对以下情况显示工作标签页和个人标签页：

- 在“设置”应用中，特别是针对“位置”、“存储”、“帐号”和“应用信息”进行显示。
- 当用户点按分享 share 时。
- 当系统向用户显示通过其他应用打开所选项目的选项时（“打开方式”菜单）。
- 选择文档时。

Android 11 还提升了用户体验，当用户的工作资料已暂停使用时，让用户能够更清楚地知道。此外，如果用户的工作密码与设备密码相同，那么当用户打开其工作资料时，不必再输入工作密码。

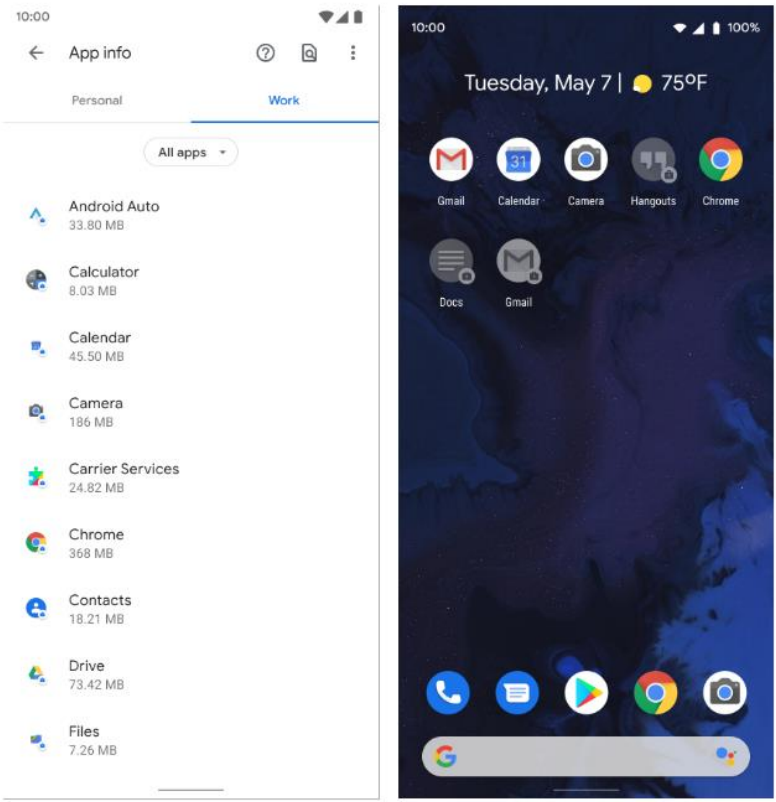


图 .（左侧）个人标签页和工作标签页（依次转到“设置”>“应用信息”）。（右侧）工作资料已暂停使用时显示的工作应用图标。

重置工作资料密码按钮

对于具有单独的设备密码和工作资料密码的 Android 11 设备，当工作资料已暂停使用时，工作资料锁定屏幕现在支持“忘记了密码”

按钮。如果 DPC 可感知直接启动，您可以设置并激活令牌以启用该按钮。

当用户按该按钮时，系统会向其显示相关文本，指示他们与 IT 管理员联系。此外，按该按钮时，还会在直接启动（锁定）模式下启动工作资料，这样可让 DPC 完成安全的工作资料密码重置的执行步骤。

B 公司自有设备

以下新功能适用于公司自有设备。“公司自有设备”一词指的是完全托管设备和归公司所有的工作资料设备。

Common Criteria 模式

此模式可以满足 Common Criteria Mobile Device Fundamentals Protection Profile (MDFPP) 的具体要求。公司自有设备的管理员现在可以在设备上启用 Common Criteria 模式（并检查该模式是否已启用）。启用后，Common Criteria 模式可以增强设备上某些安全组件的安全性，包括蓝牙长期密钥的 AES-GCM 加密和 Wi-Fi 配置存储。

单个密钥认证支持

★ 注意：此功能仅适用于配备 StrongBox 安全芯片 的设备。

在 Android 11 中，公司自有设备的管理员可以使用单个认证证书请求设备认证：

- 确保 KeyGenParameterSpec 是使用指定 StrongBox 构建的。
- 对参数 idAttestationFlags，传递 ID_TYPE_INDIVIDUAL_ATTESTATION。

还可以使用新增的方法来检查设备是否支持唯一设备 ID 认证。

C 其他

现在，当管理员执行以下操作时，用户会收到通知：

- 在公司自有设备上启用位置信息服务。如果管理员设置了一项全局政策以自动接受所有权限，则当应用请求位置信息权限并因这项政策而被授予该权限时，用户会收到通知。
- 向应用授予权限，准许其使用个人所有设备的位置信息。

向工作应用预先授予证书访问权限：以 Android 11 为目标平台的 DPC 现在可以选择授予各个应用对特定 KeyChain 密钥的访问权限，允许这些应用直接调用 getCertificateChain() 和 getPrivateKey()，而不必先调用 choosePrivateKeyAlias()。

例如，作为后台服务运行的 VPN 应用可以使用此功能获取对所需证书的访问权限，而无需任何用户交互。还可以使用一个新方法撤消访问权限。

> ★ **注意：**安装在非托管设备上或设备的个人资料中的应用不能再使用 createInstallIntent() 安装 CA 证书，而是必须由用户在**设置**中手动安装 CA 证书。

与设置密码最低要求相关的所有方法都需要相应的密码质量才能强制执行。

- setPasswordMinimumLength() 至少需要 PASSWORD_QUALITY_NUMERIC。
- 所有其他密码最低要求方法至少需要 PASSWORD_QUALITY_COMPLEX。

始终开启的 VPN 增强功能：如果始终开启的 VPN 由管理员配置，用户就不能再停用该功能。

对 ADMIN_POLICY_COMPLIANCE 进行了更新：

- 在配置 Android 11 设备时，系统现在会先发送 ADMIN_POLICY_COMPLIANCE，然后再将 DEVICE_PROVISIONED 设置为 true。
- 在添加 Google 帐号时还可以选择使用 ADMIN_POLICY_COMPLIANCE 配置设备。在 2021 年的 Android 版本中，此配置方法是必需的。

此外，还提供了用于以下用途的新 API：

- 检查设备上是否启用了自动对时功能并进行相关设置。启用后，系统会自动从网络获取时间。取代了 setAutoTimeRequired() 和 getAutoTimeRequired()（请参阅弃用部分了解详情）。
- 检查设备上是否启用了自动时区功能并进行相关设置。启用后，系统会自动从网络获取时区。
- 检查并设置公司自有设备上的恢复出厂设置保护（FRP）政策。
- 检查用户是否可以在公司自有设备上更改管理员配置的网络设置并进行相关设置。
- 检查并设置完全托管设备上的受保护软件包。用户无法清除应用数据或强行停止受保护的软件包。
- 设置设备上的主要位置信息设置。

D 弃用

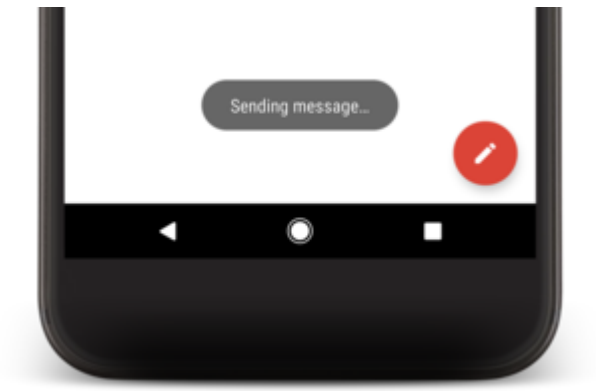
Android 11 弃用了以下 API，值得引起注意。

- resetPassword() 现已完全弃用。所有 DPC 都应该使用安全密码重置。
- setAutoTimeRequired() 和 getAutoTimeRequired()。请改用 setAutoTime() 和 getAutoTime()。
- setStorageEncryption 和 getStorageEncryption()。请改用 getStorageEncryptionStatus()。
- setGlobalSetting() 和 setSecureSetting() 大部分都已弃用 - 提供了专用的 setter 方法和用户限制来替换大多数设置（如需了解详情，请参阅参考文档）。
- setOrganizationColor() 已完全弃用。

1.5.7 消息框

消息框可以在一个小型弹出式窗口中提供与操作有关的简单反馈。它仅会填充消息所需的空间大小，并且当前 Activity 会一直显示及供用户与之互动。超时后，消息框会自动消失。

例如，点按电子邮件中的发送会触发“正在发送电子邮件…”消息框，如下面的屏幕截图所示：



如果需要用户对状态消息做出响应，请考虑使用通知。

A 基本功能

首先，使用 makeText() 方法之一实例化 Toast 对象。 此方法采用三个参数：应用 Context、文字消息和消息框时长。它会返回一个正确初始化的 Toast 对象。您可以使用 show() 显示消息框通知，如以下示例所示：

KOTLIN

```
val text = "Hello toast!"
val duration = Toast.LENGTH_SHORT

val toast = Toast.makeText(applicationContext, text, duration)
toast.show()
```

JAVA

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

此示例演示了大多数消息框通知所需的一切。几乎不需要其他内容。但是，您可能需要以不同的方式放置消息框，甚至使用您自己的布局而不是简单的文字消息。下文介绍了如何执行这些操作。

您还可以链接方法，避免保留 Toast 对象，如下所示：

KOTLIN

```
Toast.makeText(context, text, duration).show()
```

JAVA

```
Toast.makeText(context, text, duration).show();
```

B 放置消息框

标准消息框通知在屏幕底部附近水平居中显示。您可以使用 setGravity(int, int, int) 方法更改此位置。此方法接受三个参数：Gravity 常量、x 位置偏移和 y 位置偏移。

例如，如果您决定消息框应该显示在左上角，您可以按如下所示设置重心：

KOTLIN

```
toast.setGravity(Gravity.TOP or Gravity.LEFT, 0, 0)
```

JAVA

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

如果要向右移动位置，请增大第二个参数的值。要向下移动，请增大最后一个参数的值。

C

创建自定义消息框视图

如果简单的文字消息不足够，您可以为消息框通知创建自定义布局。要创建自定义布局，请在 XML 或应用代码中定义 View 布局，并将根 View 对象传递给 setView(View) 方法。

以下代码段包含消息框通知的自定义布局（另存为 layout/custom_toast.xml）：

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
              android:id="@+id/custom_toast_container"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:padding="8dp"
              android:background="#DAAA"
              >
    <ImageView android:src="@drawable/droid"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:layout_marginRight="8dp"
               />
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:textColor="#FFF"
              />
```

```
    </LinearLayout>
```

请注意，LinearLayout 元素的 ID 是“custom_toast_container”。您必须使用此 ID 和 XML 布局文件“custom_toast”的 ID 扩充布局，如下所示：

KOTLIN

```
    val inflater = layoutInflater
    val container: ViewGroup =
findViewById(R.id.custom_toast_container)
    val layout: ViewGroup = inflater.inflate(R.layout.custom_toast,
container)
    val text: TextView = layout.findViewById(R.id.text)
    text.text = "This is a custom toast"
    with (Toast(applicationContext)) {
        setGravity(Gravity.CENTER_VERTICAL, 0, 0)
        duration = Toast.LENGTH_LONG
        view = layout
        show()
    }
```

JAVA

```
    LayoutInflater inflater = getLayoutInflater();
```

```
    View layout = inflater.inflate(R.layout.custom_toast,
                    (ViewGroup)
findViewById(R.id.custom_toast_container));

    TextView text = (TextView) layout.findViewById(R.id.text);
    text.setText("This is a custom toast");

    Toast toast = new Toast(getApplicationContext());
    toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
    toast.setDuration(Toast.LENGTH_LONG);
    toast.setView(layout);
    toast.show();
```

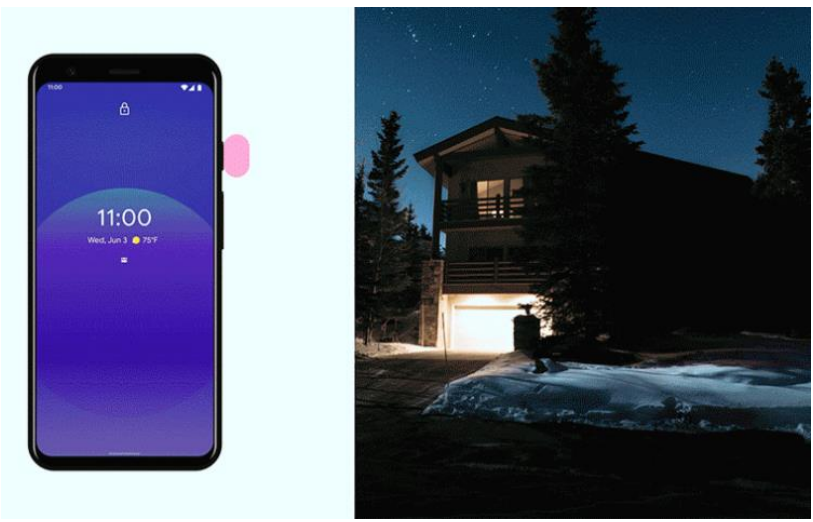
首先，使用 getLayoutInflater()（或 getSystemService()）检索 LayoutInflater，然后使用 inflate(int, ViewGroup) 扩充 XML 中的布局。第一个参数是布局资源 ID，第二个参数是根视图。您可以使用此扩充后的布局在布局中查找更多 View 对象，因此现在可以捕获并定义 ImageView 和 TextView 元素的内容。最后，使用 Toast(Context) 创建一个新消息框，并设置消息框的一些属性（例如重心和时长）。接着再调用 setView(View) 并向其传递扩充后的布局。现在，您可以通过调用 show() 使用自定义布局显示消息框。

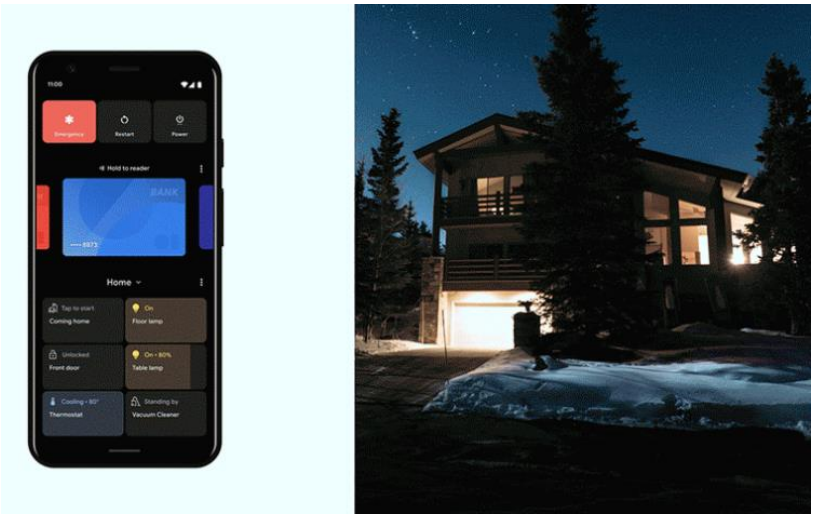
注意：除非您要使用 setView(View) 定义布局，否则请勿将公开构造函数用于消息框。如果您没有要使用的自定义布局，则必须使用 makeText(Context, int, int) 创建消息框。

1.5.8 控制外部设备

在 Android 11 及更高版本中，“快速访问设备控制器”功能可让用户通过 Android 电源菜单快速查看和控制外部设备（例如灯、恒温器和相机）。设备集合商家（例如 Google Home）和第三方供应商应用可以提供需在此工作区中显示的设备。本指南将向您介绍此工作区中设备控制器的布局，以及将其关联至控制应用的方式。

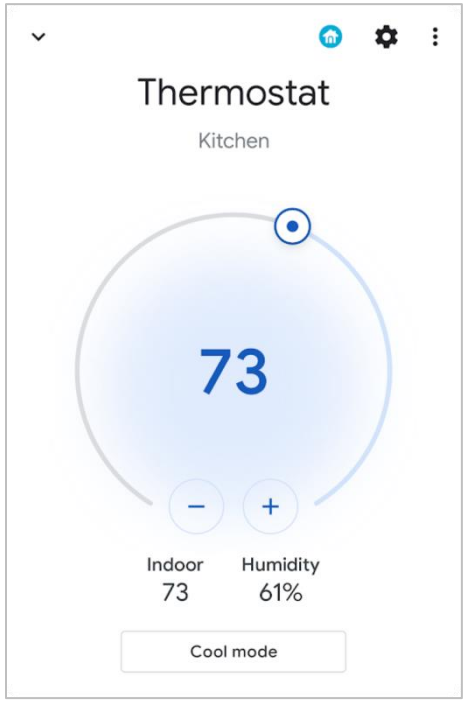
如需添加此支持，请创建并声明 ControlsProviderService，根据预定义的控件类型创建应用支持的控件，然后为这些控件创建发布者。





A 界面

设备以模板化微件的形式显示在**设备控制器**下。有五个不同的设备控制器微件可供使用：



温度面板（已打开）

长按某个微件可以转到相应的应用，让您进行更多控制操作。您可以自定义每个微件的图标和颜色，但为了提供最佳用户体验，建议您使用默认的图标和颜色，除非默认设置与设备不匹配。

B 创建服务

本部分介绍如何创建 ControlsProviderService。此服务告知 Android 系统界面，您的应用包含的设备控制器应显示在 Android 界面的**设备控制器**区域中。

ControlsProviderService API 假定您熟悉响应式流 GitHub 项目中定义的响应式流以及在 Java 9 流界面中实现的响应式流。该 API 围绕以下概念构建而成：

- 发布者：您的应用是发布者
- 订阅者：系统界面是订阅者，可以向发布者请求多个控件
- 订阅：发布者可以向系统界面发送更新的时间范围；发布者或订阅者可以关闭这个时间范围

声明服务

您的应用必须在其应用清单中声明服务。确保包含 BIND_CONTROLS 权限。

服务必须包含 ControlsProviderService 的 intent 过滤器。此过滤器能够让应用控制系统界面。

```
<!-- New signature permission to ensure only systemui can bind to these
services -->
<service                              android:name="YOUR-SERVICE-NAME"
android:label="YOUR-SERVICE-LABEL"
    android:permission="android.permission.BIND_CONTROLS">
    <intent-filter>
                                                          <action
android:name="android.service.controls.ControlsProviderService" />
    </intent-filter>
</service>
```

选择正确的控件类型

API 提供了用于创建控件的构建器方法。如需填充构建器，您需要确定要控制的设备以及用户与其互动的方式。具体而言，您需要执行以下操作：

1. 选择控件代表的设备类型。DeviceTypes 类是对当前支持的所有设备的枚举。类型用于确定设备在界面中的图标和颜色。

2. 确定面向用户显示的名称、设备所在位置（例如厨房）以及与控件关联的其他界面文本元素。

3. 选择最佳模板，为用户互动提供支持。系统会为控件分配一个来自应用的 ControlTemplate。此模板会直接向用户显示控件状态以及可用的输入方法（即 ControlAction）。下表概述了一些可用的模板及其支持的操作：

模板	操作	说明
ControlTemplate.getNoTemplateObject()	None	应用可以使用此模板来传递有关该控件的信息，但用户无法与其进行互动。
ToggleTemplate	BooleanAction	表示可在启用和停用状态之间切换的控件。BooleanAction 对象包含一个字段，当用户轻触控件时，该字段会更改以表示请求的新状态。
RangeTemplate	FloatAction	表示指定了最小值、最大值和步长值的滑块微件。当用户与滑块互动时，系统应将新的 FloatAction 对象发送回具有更新值的应用。
ToggleRangeTemplate	BooleanAction, FloatAction	此模板是 ToggleTemplate 和 RangeTemplate 的组合。它支持触摸事件和滑块，例如，在控制可调光灯的控件中。
TemperatureControlTemplate	ModeAction, BooleanAction, FloatAction	除了封装上述其中一项操作之外，此模板还允许用户设置模式，如制暖、制冷、适温、节能或关闭。
StatelessTemplate	CommandAction	用于指示提供触摸功能但无法确定其状态的控件，例如红外线电视遥控器。您可以使用此模板定义常规操作或宏，其中聚合了控件和状态更改。

了解这些信息后，现在您可以创建控件：

- 如果控件的状态未知，就使用 Control.StatelessBuilder 构建器类。
- 如果控件的状态已知，就使用 Control.StatefulBuilder 构建器类。

为控件创建发布者

创建控件后，该控件需要一个发布者。发布者将告知系统界面该控件的存在。ControlsProviderService 类有两种您必须在应用代码中替换的发布者方法：

- createPublisherForAllAvailable()：为应用中提供的所有控件创建一个 Publisher。使用 Control.StatelessBuilder() 为该发布者构建 Controls。
- createPublisherFor()：为一组给定的控件创建一个 Publisher，每个控件由其字符串标识符标识。使用 Control.StatefulBuilder 构建这些 Controls，因为发布者必须为每个控件分配一个状态。

创建发布者

当您的应用首次将控件发布到系统界面时，应用不知道每个控件的状态。获取状态可能是一项非常耗时的操作，涉及设备提供商网络中的许多跃点。使用 createPublisherForAllAvailable() 方法将可用

的控件告知系统。请注意，此方法使用 Control.StatelessBuilder 构建器类，因为每个控件的状态都是未知的。

控件显示在 Android 界面中后，用户就可以选择感兴趣的控件（即选择收藏夹）了。

注意：以下示例使用的是 RxJava 库，但您可以使用与 ReactiveStreams API 规范兼容的任何 API。

KOTLIN

```
/* If you choose to use Reactive Streams API, you will need to put the
following
 * into your module's build.gradle file:
 * implementation 'org.reactivestreams:reactive-streams:1.0.3'
 * implementation 'io.reactivex.rxjava2:rxjava:2.2.0'
 */
class MyCustomControlService : ControlsProviderService() {

    override fun createPublisherForAllAvailable(): Flow.Publisher {
        val context: Context = baseContext
        val i = Intent()
        val pi =
            PendingIntent.getActivity(
                context, CONTROL_REQUEST_CODE, i,
                PendingIntent.FLAG_UPDATE_CURRENT
            )
        val controls = mutableListOf()
```

```
        val control =
            Control.StatelessBuilder(MY-UNIQUE-DEVICE-ID, pi)
                // Required: The name of the control
                .setTitle(MY-CONTROL-TITLE)
                // Required: Usually the room where the control is located
                .setSubtitle(MY-CONTROL-SUBTITLE)
                // Optional: Structure where the control is located, an
example would be a house
                .setStructure(MY-CONTROL-STRUCTURE)
                // Required: Type of device, i.e., thermostat, light, switch
                .setDeviceType(DeviceTypes.DEVICE-TYPE) // For example,
DeviceTypes.TYPE_THERMOSTAT
                .build()
        controls.add(control)
        // Create more controls here if needed and add it to the ArrayList

        // Uses the RxJava 2 library
        return
FlowAdapters.toFlowPublisher(Flowable.fromIterable(controls))
    }
}
```

JAVA

```
/* If you choose to use Reactive Streams API, you will need to put the
following
 * into your module's build.gradle file:
 * implementation 'org.reactivestreams:reactive-streams:1.0.3'
 * implementation 'io.reactivex.rxjava2:rxjava:2.2.0'
```

```
 */
public class MyCustomControlService extends ControlsProviderService {

    @Override
    public Publisher createPublisherForAllAvailable() {
        Context context = getBaseContext();
        Intent i = new Intent();
        PendingIntent pi = PendingIntent.getActivity(context, 1, i,
PendingIntent.FLAG_UPDATE_CURRENT);
        List controls = new ArrayList<>();
        Control control = new
Control.StatelessBuilder(MY-UNIQUE-DEVICE-ID, pi)
          // Required: The name of the control
          .setTitle(MY-CONTROL-TITLE)
          // Required: Usually the room where the control is located
          .setSubtitle(MY-CONTROL-SUBTITLE)
          // Optional: Structure where the control is located, an example
would be a house
          .setStructure(MY-CONTROL-STRUCTURE)
          // Required: Type of device, i.e., thermostat, light, switch
          .setDeviceType(DeviceTypes.DEVICE-TYPE) // For example,
DeviceTypes.TYPE_THERMOSTAT
          .build();
        controls.add(control);
        // Create more controls here if needed and add it to the ArrayList

        // Uses the RxJava 2 library
        return
FlowAdapters.toFlowPublisher(Flowable.fromIterable(controls));
```

```
    }
}
```

用户选择一组控件后，请仅为这些控件创建发布者。使用 createPublisherFor() 方法，原因是此方法使用 Control.StatefulBuilder 构建器类，该类提供每个控件的当前状态（例如，启用或停用）。

KOTLIN

```
class MyCustomControlService : ControlsProviderService() {
    private lateinit var updatePublisher: ReplayProcessor

    override fun createPublisherFor(controlIds: MutableList):
Flow.Publisher {
        val context: Context = baseContext
        /* Fill in details for the activity related to this device. On long
press,
         * this Intent will be launched in a bottomsheet. Please design the
activity
         * accordingly to fit a more limited space (about 2/3 screen
height).
         */
        val i = Intent(this, CustomSettingsActivity::class.java)
        val pi =
            PendingIntent.getActivity(context,
CONTROL_REQUEST_CODE, i, PendingIntent.FLAG_UPDATE_CURRENT)
        updatePublisher = ReplayProcessor.create()
```

```
        if (controlIds.contains(MY-UNIQUE-DEVICE-ID)) {
            val control =
                Control.StatefulBuilder(MY-UNIQUE-DEVICE-ID, pi)
                    // Required: The name of the control
                    .setTitle(MY-CONTROL-TITLE)
                    // Required: Usually the room where the control is
located
                    .setSubtitle(MY -CONTROL-SUBTITLE)
                    // Optional: Structure where the control is located, an
example would be a house
                    .setStructure(MY-CONTROL-STRUCTURE)
                    // Required: Type of device, i.e., thermostat, light,
switch
                    .setDeviceType(DeviceTypes.DEVICE-TYPE) // For
example, DeviceTypes.TYPE_THERMOSTAT
                    // Required: Current status of the device
                    .setStatus(Control.CURRENT-STATUS) // For example,
Control.STATUS_OK
                    .build()

            updatePublisher.onNext(control)
        }

        // If you have other controls, check that they have been selected
here

        // Uses the Reactive Streams API
        updatePublisher.onNext(control)
    }
```

```
}
```

JAVA

```
 private ReplayProcessor updatePublisher;

@Override
public Publisher createPublisherFor(List controlIds) {

    Context context = getBaseContext();
    /* Fill in details for the activity related to this device. On long press,
     * this Intent will be launched in a bottomsheet. Please design the
activity
     * accordingly to fit a more limited space (about 2/3 screen height).
     */
    Intent i = new Intent();
    PendingIntent pi = PendingIntent.getActivity(context, 1, i,
PendingIntent.FLAG_UPDATE_CURRENT);

    updatePublisher = ReplayProcessor.create();

    // For each controlId in controlIds

    if (controlIds.contains(MY-UNIQUE-DEVICE-ID)) {

      Control control = new
Control.StatefulBuilder(MY-UNIQUE-DEVICE-ID, pi)
        // Required: The name of the control
        .setTitle(MY-CONTROL-TITLE)
```

```
            // Required: Usually the room where the control is located
            .setSubtitle(MY-CONTROL-SUBTITLE)
            // Optional: Structure where the control is located, an example
would be a house
            .setStructure(MY-CONTROL-STRUCTURE)
            // Required: Type of device, i.e., thermostat, light, switch
            .setDeviceType(DeviceTypes.DEVICE-TYPE) // For example,
DeviceTypes.TYPE_THERMOSTAT
            // Required: Current status of the device
            .setStatus(Control.CURRENT-STATUS) // For example,
Control.STATUS_OK
            .build();

        updatePublisher.onNext(control);
    }
    // Uses the Reactive Streams API
    return FlowAdapters.toFlowPublisher(updatePublisher);
}
```

处理操作

performControlAction() 方法会发出信号，表明用户已与已发布的控件进行了互动。操作由已发送的 ControlAction 类型指定。对给定的控件执行适当的操作，然后在 Android 界面中更新设备的状态。

KOTLIN

```
class MyCustomControlService : ControlsProviderService() {

    override fun performControlAction(
        controlId: String, action: ControlAction, consumer: Consumer) {

        /* First, locate the control identified by the controlId. Once it is
located, you can
         * interpret the action appropriately for that specific device. For
instance, the following
         * assumes that the controlId is associated with a light, and the
light can be turned on
         * or off.
         */
        if (action is BooleanAction) {

            // Inform SystemUI that the action has been received and is
being processed
            consumer.accept(ControlAction.RESPONSE_OK)

            // In this example, action.getNewState() will have the
requested action: true for “On”,
            // false for “Off”.

            /* This is where application logic/network requests would be
invoked to update the state of
             * the device.
             * After updating, the application should use the publisher to
update SystemUI with the new
```

```
                * state.
                */
               Control control = Control.StatefulBuilder
(MY-UNIQUE-DEVICE-ID, pi)
               // Required: The name of the control
                 .setTitle(MY-CONTROL-TITLE)
                 // Required: Usually the room where the control is located
                 .setSubtitle(MY-CONTROL-SUBTITLE)
                 // Optional: Structure where the control is located, an
example would be a house
                 .setStructure(MY-CONTROL-STRUCTURE)
                 // Required: Type of device, i.e., thermostat, light, switch
                 .setDeviceType(DeviceTypes.DEVICE-TYPE) // For example,
DeviceTypes.TYPE_THERMOSTAT
                 // Required: Current status of the device
                 .setStatus(Control.CURRENT-STATUS) // For example,
Control.STATUS_OK
                 .build()

               // This is the publisher the application created during the call
to createPublisherFor()
               updatePublisher.onNext(control)
           }
       }
}
```

JAVA

```
@Override
```

```
public void performControlAction(String controlId, ControlAction
action,
    Consumer consumer) {

  /* First, locate the control identified by the controlId. Once it is
located, you can
   * interpret the action appropriately for that specific device. For
instance, the following
   * assumes that the controlId is associated with a light, and the light
can be turned on
   * or off.
   */
  if (action instanceof BooleanAction) {

    // Inform SystemUI that the action has been received and is being
processed
    consumer.accept(ControlAction.RESPONSE_OK);

    BooleanAction action = (BooleanAction) action;
    // In this example, action.getNewState() will have the requested
action: true for “On”,
    // false for “Off”.

    /* This is where application logic/network requests would be
invoked to update the state of
     * the device.
     * After updating, the application should use the publisher to update
SystemUI with the new
     * state.
```

```
     */
    Control control = new
Control.StatefulBuilder(MY-UNIQUE-DEVICE-ID, pi)
        // Required: The name of the control
        .setTitle(MY-CONTROL-TITLE)
        // Required: Usually the room where the control is located
        .setSubtitle(MY-CONTROL-SUBTITLE)
        // Optional: Structure where the control is located, an example
would be a house
        .setStructure(MY-CONTROL-STRUCTURE)
        // Required: Type of device, i.e., thermostat, light, switch
        .setDeviceType(DeviceTypes.DEVICE-TYPE) // For example,
DeviceTypes.TYPE_THERMOSTAT
        // Required: Current status of the device
        .setStatus(Control.CURRENT-STATUS) // For example,
Control.STATUS_OK
        .build();

    // This is the publisher the application created during the call to
createPublisherFor()
    updatePublisher.onNext(control);
  }
}
```

1.5.9 将自动填充功能与键盘集成

从 Android 11 开始，键盘及其他输入法（IME）可以在建议栏或类似的容器中以内嵌方式显示自动填充建议，而不是系统在下拉菜单中显示这些建议。由于这些自动填充建议可能包含隐私数据（如密码或信用卡信息），因此在用户选择某条建议之前，这些建议对 IME 隐藏。IME 和密码管理器都需要更新，才能使用此功能。如果 IME 或密码管理器不支持内嵌自动填充功能，建议会显示在下拉菜单中，就像在 Android 11 之前一样。

A 工作流程

为了理解内嵌自动填充功能的工作原理，将整个过程过一遍会很有帮助。在此流程中，IME 表示当前的键盘或其他输入法，建议提供程序表示该自动填充建议的相应提供程序。根据输入字段和用户的设置，建议提供程序可能是平台或自动填充服务。

1. 用户将焦点置于一个会触发自动填充功能的输入字段上，如密码或信用卡输入字段。
2. 平台查询当前的 IME 和相应的建议提供程序，查看它们是否支持内嵌自动填充功能。如果 IME 或建议提供程序不支持内嵌自动填充功能，建议会显示在下拉菜单中，就像在 Android 10 及更低版本中一样。

3. 平台要求 IME 提供建议请求。此建议请求指定了 IME 需要的最大建议数量，还提供了每条建议的呈现规范。呈现规范指定了最大大小、文字大小、颜色和字体数据等等，从而让建议提供程序能够匹配 IME 的外观和风格。

4. 平台要求建议提供程序提供建议，不超过请求的数量。每条建议都包含一个回调，以扩充包含建议界面的 View。

5. 平台通知 IME 建议已准备就绪。IME 调用回调方法以扩充每条建议的 View，从而显示建议。为了保护用户的隐私信息，IME 在这一阶段看不到建议是什么。

6. 如果用户选择了其中一条建议，系统会通知 IME，通知方法与用户从下拉列表中选择建议时一样。

下面几部分介绍了如何配置 IME 或密码管理器以支持内嵌自动填充功能。

B 配置 IME 以支持内嵌自动填充功能

本部分介绍了如何配置 IME 以支持内嵌自动填充功能。如果 IME 不支持内嵌自动填充功能，平台会默认在下拉菜单中显示自动填充建议。

IME 必须将 supportsInlinedSuggestions 属性设为 true：

```
<input-method
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:supportsInlineSuggestions="true"/>
```

当平台需要自动填充建议时，它会调用 IME 的 InputMethodService.onCreateInlineSuggestionsRequest() 方法。您必须实现此方法。返回 InlineSuggestionsRequest，该对象指定了以下内容：

- IME 需要多少条建议
- 每条建议的 InlinePresentationSpec，用于定义应如何呈现建议

> ★ **注意**：如果您提供的呈现规范少于请求的建议数量，会将最后一条规范用于所有多余的建议。这意味着，例如，如果您只提供一条呈现规范，建议提供程序会将该规范用于所有建议。

平台有了建议后，它会调用 IME 的 onInlineSuggestionsResponse() 方法，并传递包含建议的 InlineSuggestionsResponse。您必须实现此方法。您的实现会调用 InlineSuggestionsResponse.getInlineSuggestions() 以获取建议的列表，然后通过调用 InlineSuggestion.inflate() 方法来扩充每条建议。

C 配置自动填充服务以支持内嵌自动填充功能

本部分介绍了如何配置密码管理器以支持内嵌自动填充功能。如果您的应用不支持内嵌自动填充功能，平台会默认在下拉菜单中显示自动填充建议。

密码管理器必须将 supportsInlinedSuggestions 属性设为 true：

```
<autofill-service
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:supportsInlineSuggestions="true"/>
```

当 IME 需要自动填充建议时，平台会调用您的自动填充服务的 onFillRequest() 方法，就像在 Android 11 之前一样。不过，您的服务必须调用传递的 FillRequest 对象的 getInlineSuggestionsRequest() 方法，以获取 IME 创建的 InlineSuggestionsRequest。InlineSuggestionsRequest 指定了需要多少条内嵌建议，以及应如何呈现每条建议。如果 IME 不支持内嵌建议，该方法会返回 null。

您的自动填充服务会创建 InlinePresentation 对象，不超过 InlineSuggestionsRequest 中请求的最大数量。呈现方式必须遵循由 InlineSuggestionsRequest 指定的大小约束条件。为了将建议返回给 IME，请针对每条建议调用一次 Dataset.Builder.setValue()。Android 11 提供了新版本的 Dataset.Builder.setValue() 来支持内嵌建议。

注意：虽然 IME 应采用您的服务提供的建议，但不能保证它一定会这样做。

返回主目录

第二章

Android 11 使用入门

目录

返回主目录

2.1 设置 Android 11 SDK

返回本章目录

Android 11 可让您通过各种绝佳方式扩展应用，还包含旨在延长电池续航时间、提升安全性和增强用户隐私保护的行为变更。其中一些行为变更仅影响以 Android 11 为目标平台的应用，而其他变更会影响在 Android 11 设备上运行的所有应用（无论应用的 targetSDKVersion 为何）。

如需使用 Android 11 API 进行开发并根据 Android 11 行为变更测试应用，请按照此页面中的说明在 Android Studio 中设置 Android 11 SDK，并在 Android 11 上构建和运行应用。

2.1.1 获取最新的 Android Studio 预览版

Android 11 SDK 包含一些与某些旧版 Android Studio 不兼容的变更。因此，为了获得最佳的 Android 11 SDK 开发体验，我们建议您安装最新的 Android Studio 预览版。

获取 ANDROID STUDIO 预览版

您可以使用 Android Studio 3.3 及更高版本编译和测试 Android 11 应用，但部分 Android 11 SDK 用户可能会遇到 Gradle 同步失败问题以及与过时依赖项有关的警告。请注意，您可以保留已安装的现有 Android Studio 版本，因为您可以并行安装多个版本。

2.1.2 获取 Android 11 SDK

安装并打开 Android Studio 后，请按以下步骤安装 Android 11 SDK：

7. 依次点击 Tools > SDK Manager。
8. 在 SDK Platforms 标签页中，选择 Android 11。
9. 在 SDK Tools 标签页中，选择 Android SDK Build-Tools 30（或更高版本）。
10. 点击 OK 开始安装。

2.1.3 更新构建配置

将应用的构建配置更改为以 Android 11 为目标平台可让您的应用访问 Android 11 API，并让您在准备添加对 Android 11 的全面支持时全面测试您的应用的兼容性。如需进行此更改，请打开模块级 build.gradle 文件并更新 compileSdkVersion 和 targetSdkVersion：

```
android {
    compileSdkVersion 30

    defaultConfig {
        targetSdkVersion 30
    }
    ...
}
```

如需了解 Android 11 中可能影响您的应用的变更以便开始对其进行测试，请参阅以下页面：

- 影响所有应用的 Android 11 行为变更
- 影响以 Android 11 为目标平台的应用的 Android 11 行为变更
- Android 11 隐私权变更

如需详细了解 Android 11 中提供的新 API，请参阅 Android 11 功能和 API。

2.2

获取 Android 11

返回本章目录

您可以通过以下任一种方式获取 Android 11：

1. 获取适用于 Google Pixel 设备的 OTA 更新或系统映像
2. 设置 Android 模拟器以运行 Android 11
3. 获取适用于符合 Treble 标准的合格设备的 GSI 系统映像

如需了解如何设置 Android Studio 以进行测试和开发，请参阅设置 SDK。

2.2.1 在 Pixel 设备上获取 Android 11

如果您使用的是合格的 Google Pixel 设备，则可以检查并更新 Android 版本，以便通过无线下载的方式接收 Android 11。

另外，如果您希望手动刷写设备，则可以在 Pixel 下载页面上获取设备的 Android 11 系统映像。请阅读有关如何将系统映像刷写到设备上的一般说明。当您需要更好地控制测试（例如自动测试或回归测试）时，此方法可能非常实用。

在大多数情况下，您无需重置所有数据即可切换到 Android 11，但建议您在注册设备前对数据进行备份。

Android 11 适用于 Pixel 3/3a 和 3/3a XL、Pixel 2 和 2 XL 以及 Pixel 和 Pixel XL。

2.2.2 设置 Android 模拟器以运行 Android 11

配置 Android 模拟器以运行 Android 11 是探索新功能和 API 以及测试 Android 11 行为变更的理想解决方案。设置模拟器既快捷又方便，可让您模拟各种屏幕尺寸和设备特性。

您可以在 Android Studio 中使用 Android 11 设置模拟器：

1. 安装 Android Studio 的最新预览版 build。
2. 在 Android Studio 中，依次点击 **Tools > SDK Manager**。
3. 在 **SDK Tools** 标签页中，选择最新版 **Android 模拟器**，然后点击 **OK**。系统会安装最新版本（如果尚未安装）。

在 Android Studio 中，依次点击 **Tools > AVD Manager**，然后按照说明创建一个新 AVD。

请务必选择 Pixel 2、Pixel 3、Pixel 3a、Pixel 4 或 Pixel 4a 设备定义以及 Android 11（API 级别 30）系统映像。如果您尚未安装与您的设备定义匹配的 Android 11 系统映像，请点击 **Release Name** 旁边的 **Download** 下载该映像。

返回到 AVD 管理器中的虚拟设备列表后，双击新虚拟设备即可启动该设备。

2.2.3 在 Android GSI 上进行测试

Android 通用系统映像 (GSI) 二进制文件可供开发者在受支持且符合 Treble 标准的设备上进行应用测试和验证。在 Android 11 正式发布之前，您可以使用这些映像解决 Android 11 的所有兼容性问题，并发现和报告操作系统和框架方面的问题。

如需了解设备要求、刷机说明以及有关为设备选择正确映像类型的信息，请参阅 GSI 文档。准备好下载 GSI 二进制文件后，请参阅 Android 11 GSI 页面上的“下载”部分。

返回主目录

第三章

Android 11 迁移指南

目录

返回主目录

迁移前请先了解！

每次发布新的 Android 版本时，我们都会推出一些全新的功能并引入一些行为变更，目的就在于提高 Android 的实用性、安全性和性能。在许多情况下，您的应用都可以直接使用并完全按预期运行；而在其他的一些情况下，您可能需要对应用进行更改以适应这些平台变化。

源代码发布到 AOSP（Android 开源平台）后，用户随之就可能开始使用新平台。因此，应用必须做好准备，让用户能够正常使用，最好还能利用新的功能和 API 发挥新平台的最大优势。

本节对典型的开发和测试阶段进行了简要介绍，以帮助您制定与平台发布时间表保持一致的准备计划，并确保您的用户在 Android 11 上获得良好的体验。

典型的迁移包含两个阶段，这两个阶段可以同时进行：

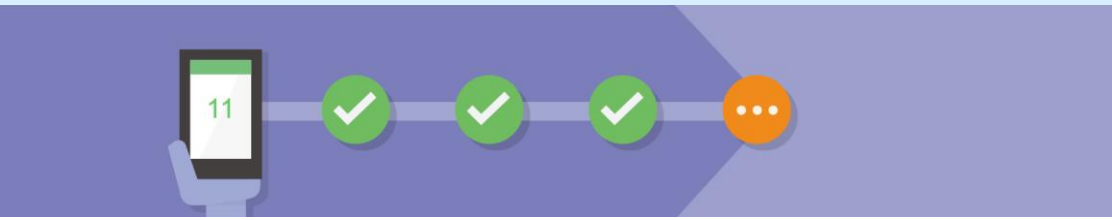
- 确保应用兼容性（在 Android 11 最终发布前）
- 针对新平台的功能和 API 调整应用（最终发布后尽快进行）

本页概述了其中的每一个阶段的一般步骤。当您准备好开始后，请阅读获取 Android 11。

返回本章目录

第 1 阶段：
确保与 Android 11 兼容

返回本章目录



本页概述了其中的每一个阶段的一般步骤。当您准备好开始后，请阅读获取 Android 11。

您必须测试现有应用在 Android 11 上的运行情况，以确保更新到最新版 Android 的用户获得良好的体验。有些平台变更可能会影响应用的行为方式，因此，必须尽早进行全面测试并对应用进行任何必要的调整。

通常，您可以调整应用并发布更新，而无需更改应用的 targetSdkVersion。同样，您应该也不需要使用新的 API 或更改应用的 compileSdkVersion，但是这一点可能要取决于应用的构建方式及其所使用的平台功能。以下是具体步骤。

即使您无需更改应用的 targetSdkVersion，在开始之前也务必要熟悉可能影响应用的行为变更。

执行兼容性测试

在大多数情况下，测试应用与 Android 11 的兼容性与您在准备发布应用时所执行的测试类似。这时有必要查阅核心应用质量指南和测试最佳做法。

只需在搭载 Android 11 的设备上安装您当前发布的应用，并完成所有流程和功能即可查找问题。为帮助您确定测试重点，请查看 Android 11 中引入的行为变更，这些变更会影响应用的功能或导致应用崩溃。尤其务必查看重要的隐私权变更，并测试为适应这些变更而实施的修复。

此外，请务必查看并测试受限非 SDK 接口的使用，并改为使用公共 SDK 或 NDK 等效项。留意突出显示这些访问权限的 logcat

警告，并使用 StrictMode 方法 detectNonSdkApiUsage() 以编程方式捕获它们。

最后，请务必完整测试应用中的库和 SDK，确保它们在 Android 11 上按预期运行，并遵循隐私权、性能、用户体验、数据处理和权限方面的最佳做法。如果您遇到问题，请尝试更新到最新版本的 SDK，或联系 SDK 开发者寻求帮助。

当您完成测试并进行更新后，我们建议您立即发布兼容的应用。这样可以尽早让您的用户测试应用，并帮助用户顺利过渡到 Android 11。

第三方兼容性测试资源

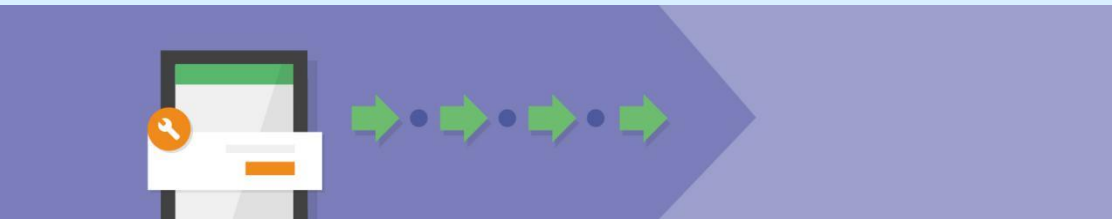
您也可以考虑借助第三方的兼容性测试资源进行兼容性测试。

目前，国内以下测试平台为大家提供 Android 11 的全面兼容性应用测试（具体免费时段由该测试平台决定）。

腾讯 WeTest Android 11 标准兼容测试平台

第 2 阶段：更新应用目标并使用新 API 进行构建

返回本章目录

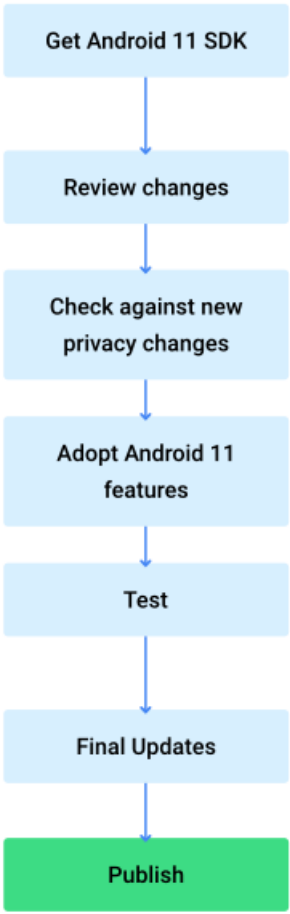


按照上述说明发布应用的兼容版本后，下一步是通过更新 targetSdkVersion 并利用 Android 11 的新 API 和功能来添加对 Android 11 的全面支持。在准备就绪后，您就可以立即执行这些操作，并牢记针对新平台的 Google Play 要求。

当您计划全面支持 Android 11 时，请先查看适用于以 Android 11 为目标平台的应用的行为变更。这些针对性的行为变更可能会导致需要解决的功能问题。在某些情况下，可能需要进行大量开发工作，因此最好尽早了解。为帮助您评估影响，您还可以在启用所选变更的情况下，使用兼容性切换开关来测试当前应用。

以下是全面支持 Android 11 的步骤。

第 2 阶段：更新应用目标并使用新 API 进行构建



Install Android Studio 3.5 or higher to build with Android 11. Make sure you have an Android 11 device or emulator. Update your `targetSdkVersion` and other build configurations.

Review the system behavior changes that apply to apps targeting Android 11. Identify areas where your app may be affected, and plan how to support them.

Make code and architecture changes needed to support Android 11's new user privacy changes.

Take advantage of Android 11 APIs to bring new features and capabilities to your apps. Recompile for Android 11.

Test on an Android 11 device or emulator. Focus on areas where behavior changes may affect your app. Exercise new APIs. Report any platform, API, or third-party SDK issues.

When Android 11 APIs are final, update your `targetSdkVersion` and other build configurations again, make any additional updates, and test your app.

Sign, upload, and publish your updated Android App Bundle or APK.

获取 SDK，更改目标平台，使用新 API 进行构建

如需开始全面支持 Android 11，请先将 Android 11 SDK（以及所需的任何其他工具）下载到 Android Studio 中。接下来，将应用的 targetSdkVersion 和 compileSdkVersion 更改为 "30"，然后重新编译应用。有关详情，请参阅设置指南。

测试 Android 11 应用

编译应用并将其安装到搭载 Android 11 的设备上后，请开始测试，以确保应用能够在 Android 11 上正常运行。某些行为变更仅在您的应用以新平台为目标平台时才适用，因此您需要在开始之前查看这些变更。

与基本兼容性测试一样，完成所有流程和功能以查找问题。将测试重点放在以 Android 11 为目标平台的应用的行为变更上。尤其务必查看隐私权变更，并测试为适应这些变更而实施的修复。您还可以根据核心应用质量指南和测试最佳做法检查您的应用。

请务必查看并测试可能适用的受限非 SDK 接口的使用。留意突出显示这些访问权限的 logcat 警告，并使用 StrictMode 方法 detectNonSdkApiUsage() 以编程方式捕获它们。

最后，请务必完整测试应用中的库和 SDK，确保它们在 Android 11 上按预期运行，并遵循隐私权、性能、用户体验、数据处理和权限方面的最佳做法。如果您遇到问题，请尝试更新到最新版本的 SDK，或联系 SDK 开发者寻求帮助。

使用应用兼容性切换开关进行测试

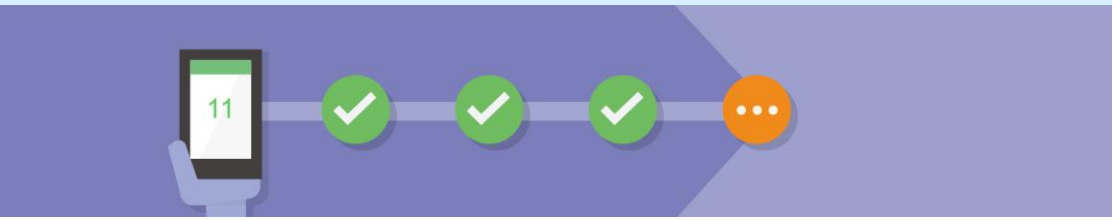
Android 11 为开发者引入了一项新功能，可让您更轻松地测试应用的针对性行为变更。对于可调试的应用，切换开关可让您：

- **在不实际更改应用的 targetSdkVersion 的情况下测试有针对性的更改**。您可以使用切换开关强制启用特定的针对性行为变更，以评估对现有应用的影响。
- **仅针对特定变更进行测试**。您可以使用切换开关停用除要测试的变更除之外的所有针对性变更，而不必一次处理所有针对性变更。
- **通过 adb 管理切换开关**。您可以使用 adb 命令在自动测试环境中启用和停用可切换的变更。
- **使用标准变更 ID 更快地进行调试**。每个可切换的变更都具有唯一 ID 和名称，可用于在日志输出中快速调试根本原因。

在您准备更改应用的目标平台时，或者在您积极开发以便支持 Android 11 时，切换开关将十分有用。请在此处填写详细信息。

测试应用与 Android 11 的兼容性

返回本章目录



Android 11 引入了一些工具，供您用于测试和调试自己的应用，检查其是否兼容最新版平台中的行为变更。这些工具属于新的兼容性框架的一部分，可让应用开发者单独开启和关闭各项变更。有了这种灵活性，您既可以关闭某一项变更，然后继续针对平台中的其他变更测试应用；也可以每次单独针对一项行为变更测试应用。

不管是影响所有应用的行为变更，还是只影响以 Android 11 为目标平台的应用的行为变更，您都可以随意开启或关闭。

A

如何识别已启用哪些变更

您可以使用开发者选项、logcat 或 ADB 命令查看当前已启用的行为变更。

使用开发者选项识别已启用的变更

您可以在设备的开发者选项中查看当前已启用的变更，并且可以开启/关闭这些变更。如需访问这些选项，请按以下步骤操作：

1. 如果开发者选项尚未启用，请启用开发者选项。
2. 打开设备的“设置”应用，然后依次导航到**系统 > 高级 > 开发者选项 > 应用兼容性变更**。
3. 从列表中选择您的应用。

每项行为变更通常属于以下两种类别之一：

- 影响在 Android 11 上运行的所有应用（无论应用的 `targetSdkVersion` 是什么）的变更。

  默认情况下，这些变更在兼容性框架中处于启用状态，并且会在界面的**默认启用的应用兼容性变更**部分列出。

- 只会影响以 Android 11 为目标平台的应用的变更。

  如果您的应用以 Android 11 为目标平台，默认情况下，这些变更在兼容性框架中处于启用状态，并且会在界面的**在 SDK 29 之后启用的应用兼容性变更**部分列出。

在图 1 中，您还会看到一个名为**默认停用的应用兼容性变更**部分。此部分中的变更通常为实验性变更。

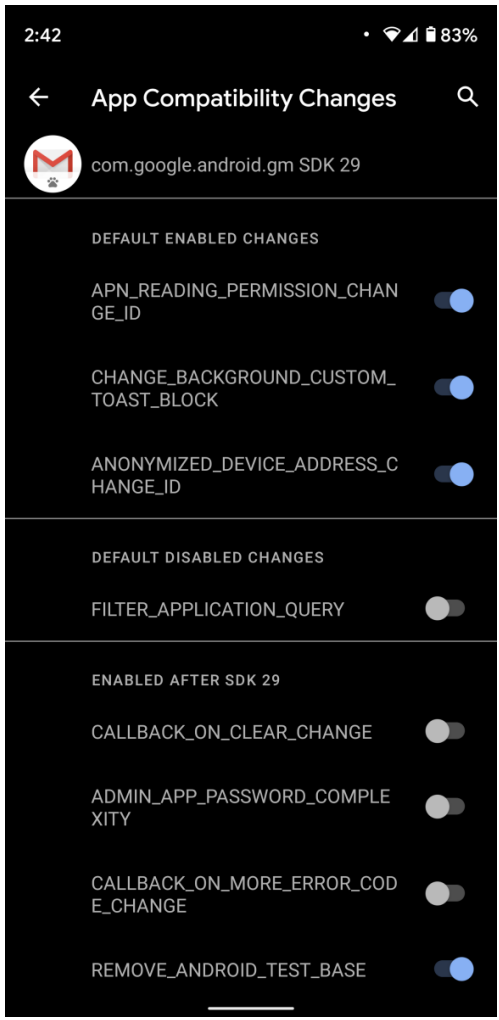


图 1. 开发者选项中的“应用兼容性变更”屏幕

使用 logcat 识别已启用的变更

对于每项行为变更，应用在其进程运行过程中首次调用受影响的 API 时，系统都会输出一条类似如下的 logcat 消息：

```
D CompatibilityChangeReporter: Compat change id reported: 141455849; UID 10265; state: ENABLED
```

★ **注意**：对于每项变更，每个进程最多只会记录一次。为确保看到所有相关的 logcat 消息，请强行停止应用进程，然后再重启该进程。

每条 logcat 消息都包含以下信息：

变更 ID

指明影响到应用的变更。该值映射到“应用兼容性变更”屏幕中列出的某项行为变更（参见图 1）。在此示例中，`141455849` 映射到 `ANONYMIZED_DEVICE_ADDRESS_CHANGE_ID`。

UID

指示受该变更影响的应用。

状态

指示该变更当前是否正在影响应用。

状态可以是以下值之一：

状态	含义
ENABLED	该变更当前已启用，如果应用使用已变更的相应 API，该变更将会影响应用的行为。
DISABLED	该变更当前已停用，不会影响应用。 注意：如果此变更停用的原因是应用的 targetSDKVersion 低于要求的阈值（应用不是以 Android 11 为目标平台），当应用提高其 targetSDKVersion，从而以 Android 11 为目标平台时，系统会默认启用此变更。
LOGGED	此变更当前通过兼容性框架记录，但无法开启或关闭。虽然无法切换此变更的状态，但它仍可能会影响应用的行为。如需了解详情，请参阅行为变更列表中的变更说明。在许多情况下，这些类型的变更是实验性变更，可以忽略。

使用 ADB 识别已启用的变更

如需查看整个设备上的所有变更（包括已启用和已停用的变更），请运行以下 ADB 命令：

```
$ adb shell dumpsys platform_compat
```

输出内容中会列出每项变更的以下信息：

变更 ID

此行为变更的唯一标识符，例如 `141455849`。

名称

此行为变更的名称，例如 `ANONYMIZED_DEVICE_ADDRESS_CHANGE_ID`。

targetSDKVersion 条件

控制此变更是否启用的 `targetSDKVersion` 值（如果有）。

例如，如果此变更仅对以 SDK 版本 30 或更高版本为目标平台的应用才会启用，系统会输出 `enableAfterTargetSdk=29`。如果此变更的启用与否不受 `targetSDKVersion` 控制，系统会输出 `enableAfterTargetSdk=0`。

软件包替换

其中变更的默认状态（启用或停用）已被替换的各个软件包的名称。

例如，如果此变更默认处于启用状态，而您使用开发者选项或 ADB 关闭了此变更，应用的软件包名称会在此列出。在这种情况下，输出结果如下：

```
packageOverrides={com.my.package=false}
```

受 `targetSDKVersion` 控制的变更在默认情况下可以处于启用状态，也可以处于停用状态，因此该软件包列表可以同时包含 true 和 false 实例，具体取决于每个应用的 `targetSDKVersion`。例如：

```
packageOverrides={com.my.package=true, com.another.package=false}
```

B 在何时切换变更的开启/关闭状态

兼容性框架的主要目的在于，在您使用新版 Android 测试应用时，为您提供可控性和灵活性。

何时关闭变更

对于特定的平台版本，影响所有应用（无论应用的 targetSDKVersion 是什么）的变更在默认情况下处于启用状态。通常，您首先需要针对这些变更测试并更新您的应用，以确保用户在该版本平台上使用应用的体验不会受到影响。您也应该优先测试这些变更，因为在使用 Android 的公开发布 build 时，为保护最终用户的安全，您无法关闭这些变更。

如果您的应用以特定的 targetSDKVersion 为目标平台，受该版本控制的任何变更在默认情况下也将处于启用状态。

由于应用可能会受到多项变更的影响，因此您可以逐项关闭这些变更。如果应用崩溃，您可以使用此方法确定是哪项平台变更造成了应用的崩溃。

何时开启变更

对于受特定 targetSDKVersion 控制的变更，当应用的目标 SDK 版本低于该版本阈值时，这些变更在默认情况下就会处于停用状态。在某些情况下，您可能需要启用这些变更。

例如，您可能需要针对下一 targetSdkVersion 中的一系列平台变更测试您的应用。您可以使用开发者选项或 ADB 命令逐个启用和测

试受 targetSDKVersion 控制的变更，而不是更改应用清单并一次性选择启用所有变更。这项附加控制可以帮助您单独测试各项变更，避免同时调试和更新应用的多个部分。

C 开启或关闭变更

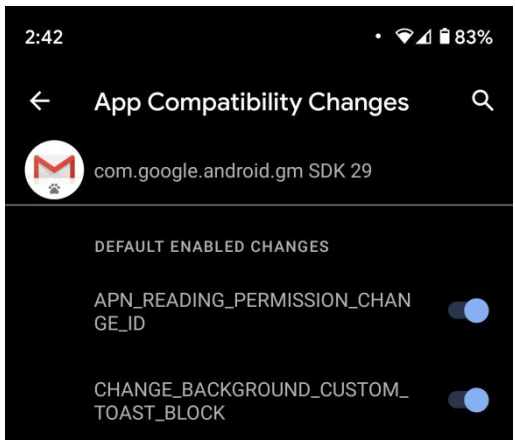
兼容性框架支持您使用开发者选项或 ADB 命令开启或关闭各项变更。由于开启或关闭变更可能会导致应用崩溃或停用重要的安全变更，因此对于何时可以切换变更的状态有一些限制。

> ★ **注意：**每次您使用开发者选项或 ADB 命令为应用开启或关闭变更时，应用都会终止，以确保您的替换操作立即生效。

使用开发者选项开启或关闭变更

您可以使用开发者选项开启或关闭变更。如需找到开发者选项，请按以下步骤操作：

1. 如果开发者选项尚未启用，请启用开发者选项。
2. 打开设备的“设置”应用，导航到**系统 > 高级 > 开发者选项 > 应用兼容性变更**。
3. 从列表中选择您的应用。
4. 在变更列表中，找到想要开启或关闭的变更，然后点按相应的开关。



使用 ADB 开启或关闭变更

如需使用 ADB 开启或关闭变更，请运行以下相应的命令：

```
$ adb shell am compat enable (CHANGE_ID|CHANGE_NAME)
PACKAGE_NAME

$ adb shell am compat disable (CHANGE_ID|CHANGE_NAME)
PACKAGE_NAME
```

您需要传递 CHANGE_ID（例如 141455849）或 CHANGE_NAME（例如 ANONYMIZED_DEVICE_ADDRESS_CHANGE_ID）以及应用的 PACKAGE_NAME。

您也可以使用以下命令将变更重置回默认状态，移除您使用 ADB 或开发者选项设置的任何替换状态：

```
$ adb shell am compat reset (CHANGE_ID|CHANGE_NAME)
PACKAGE_NAME
```

D 切换变更的开启/关闭状态的限制

默认情况下，每项行为变更不是处于启用状态，就是处于停用状态。影响所有应用的变更在默认情况下处于启用状态。其他变更的启用与否受 targetSdkVersion 控制。如果应用以相应的 SDK 版本或更高版本为目标平台，这些变更在默认情况下处于启用状态；如果应用的目标 SDK 版本低于该版本阈值，这些变更在默认情况下将处于停用状态。当您开启或关闭某项变更时，会替换其默认状态。

为了避免兼容性框架遭到恶意使用，对于何时可以切换变更的状态有一些限制。是否可以切换变更的状态取决于变更的类型、应用的可调试性，以及设备上运行的 build 类型。下表介绍了不同类型的变更的状态切换限制：

Build 类型	不可调试的应用	可调试的应用	
	所有变更	受 targetSDKVersion 控制的变更	所有其他变更
开发者预览版或 Beta build	无法切换	可以切换	可以切换
公开用户 build	无法切换	可以切换	无法切换

E 兼容性框架中包含的行为变更

此部分的列表中介绍了 Android 11 中的兼容性框架中包含的各项行为变更。您可以结合使用此列表与开发者选项和 ADB 命令对应用进行测试和调试。

> ★ **注意：**Android 11 中的某些行为变更可能尚未包含在兼容性框架中。随着我们将现有的行为变更添加到各个版本的兼容性框架中，该列表将会相应扩展。

ADD_CONTENT_OBSERVER_FLAGS

变更 ID：150939131

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的应用，系统会提供一个包含整数 `flags` 参数的新的公共 API 重载 `onChange(boolean, Uri, int)`。

对于使用包含整数 `userId` 参数的非 SDK `onChange()` 重载方法的应用，该新方法是一种公共 SDK 方法。

ADMIN_APP_PASSWORD_COMPLEXITY

变更 ID：123562444

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的管理应用，每当应用设置的密码要求与当前指定的密码质量不相符时，系统就会抛出错误。例如，当密码质量设置为 DevicePolicyManager.PASSWORD_QUALITY_UNSPECIFIED 时，应用将无法设置最小密码长度。在这种情况下，在尝试设置最小密码长度之前，应用应首先调用 setPasswordQuality() 方法，然后才能调用 setPasswordMinimumLength() 方法。

此外，如果以 Android 11 为目标平台的管理应用降低密码质量，任何不再适用的现有密码要求都会重置为其默认值。

APP_DATA_DIRECTORY_ISOLATION

变更 ID：143937733

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

以 Android 11 为目标平台的应用不能再访问任何应用的私有数据目录中的文件，而不管其他应用的目标 SDK 版本是什么。

如需了解详情，请参阅访问私有目录。

APN_READING_PERMISSION_CHANGE_ID

变更 ID：124107808

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的应用，现在必须具备 Manifest.permission.WRITE_APN_SETTINGS 权限才能访问 APN 数据库。

如需详细了解此变更，请参阅限制对 APN 数据库的读取访问。

BACKGROUND_RATIONALE_CHANGE_ID

变更 ID：147316723

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

现在，应用每次在后台请求访问设备的位置信息时，都必须提供有效的理由。

如需详细了解此变更，请参阅有关如何在 Android 11 中在后台访问位置信息的指南，该指南介绍了 Android 11 中与位置信息相关的隐私权变更。

CALLBACK_ON_CLEAR_CHANGE

变更 ID：119147584

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

现在每次调用 Editor.clear 时，都会使用 null 键回调 OnSharedPreferenceChangeListener.onSharedPreferenceChanged。

如需详细了解此变更，请参阅 OnSharedPreferenceChangeListener 的回调变更。

CALLBACK_ON_MORE_ERROR_CODE_CHANGE

变更 ID：130595455

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

错误代码现在针对 updateAvailableNetworks(List, Executor, Consumer) 和 setPreferredOpportunisticDataSubscription(int, boolean, Executor, Consumer) 进行了扩展。

CALL_BACK_ON_CHANGED_LISTENER_WITH_SWITCHED_OP_CHANGE

变更 ID：148180766

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

这是 startWatchingMode(String, String, AppOpsManager.OnOpChangedListener) 的细微行为变更。在进行此变更之前，系统会针对切换的操作回调。在进行此变更之后，系统会针对实际请求的操作回调；如果未指定操作，就会针对所有切换的操作回调。

CAMERA_MICROPHONE_CAPABILITY_CHANGE_ID

变更 ID：136219221

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的应用，只有在清单文件中将 R.attr.foregroundServiceType 分别配置为 ServiceInfo.FOREGROUND_SERVICE_TYPE_CAMERA 和 ServiceInfo.FOREGROUND_SERVICE_TYPE_MICROPHONE 时，前台服务才会接收正在使用的摄像头和麦克风功能。在较低版本的 Android 系统中，前台服务会自动接收摄像头和麦克风功能。

如需详细了解此变更，请参阅 Android 11 中的前台服务类型。

CHANGE_BACKGROUND_CUSTOM_TOAST_BLOCK

变更 ID：128611929

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

应用无法再在后台发布自定义消息框。但是，应用在后台仍然可以使用 Toast.makeText(Context, CharSequence, int) 方法及其变体发布消息框。

如需详细了解此变更，请参阅自定义消息框视图被屏蔽。

CHANGE_RESTRICT_SAW_INTENT

变更 ID：135920175

默认状态：无法切换此变更的状态。它只由兼容性框架记录。

使用 android.settings.MANAGE_APP_OVERLAY_PERMISSION 操作和 package 数据 URI 架构的 intent 不再将用户引导至应用专

用屏幕以管理相关权限。相反，用户会被引导至另一个屏幕，在该屏幕中，用户可以管理所有已请求该权限的应用。

> **注意**：此变更适用于在 Android 11 上运行的所有应用，而无论应用的 targetSdkVersion 是什么。即使您无法开启或关闭此变更，也应测试应用与此变更的兼容性。如需详细了解此变更，请参阅 MANAGE_OVERLAY_PERMISSION intent 始终会将用户转至系统权限屏幕。

CHANGE_TEXT_TOASTS_IN_THE_SYSTEM

变更 ID：147798919

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

文本消息框现在由 SystemUI 呈现，而不是在应用内呈现。这样可防止应用规避对在后台发布自定义消息框的限制。

DEFAULT_SCOPED_STORAGE

变更 ID：149924527

默认状态：对于所有应用处于启用状态。

以 Android 11 为目标平台的所有应用现在都默认使用分区存储，并且不能再停用分区存储。

不过，无论您应用的目标 SDK 版本和清单标记值是什么，您都可以在不使用分区存储的情况下测试应用，只需关闭此变更即可。

★ 注意：关闭此变更以进行应用测试时，您还必须停用 FORCE_ENABLE_SCOPED_STORAGE（如果尚未将其停用）以恢复旧存储行为。

如需详细了解 Android 11 中分区存储的变更，请转到关于 Android 11 中 Android 存储变更的页面，请参阅分区存储部分。

EMPTY_INTENT_ACTION_CATEGORY

变更 ID：151163173

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的应用，如果 intent 过滤器的 action 或 category 为空字符串，系统现在会抛出错误。Android 11 之前的平台中有一个错误，允许系统在这种情况下继续运行，而不抛出错误。请注意，这不包括相应属性为 null 或缺失的情况，因为在此类情况下始终都会抛出错误。

FILTER_APPLICATION_QUERY

变更 ID：135549675

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

应用现在需要先声明想要使用的软件包和 intent，然后才能获取有关设备上其他应用的详细信息。应用必须在其清单中使用 <queries> 标记进行声明。

如需详细了解如何在 Android 11 中查询安装的其他应用并与之交互，请参阅软件包可见性隐私权页面。

FORCE_ENABLE_SCOPED_STORAGE

变更 ID：值 - 132649864

默认状态：对于所有应用处于停用状态。

以 Android 11 为目标平台的所有应用现在都默认使用分区存储，并且不能再停用分区存储。

但是，如果您的应用仍以 Android 10（API 级别 29）或更低版本为目标平台，无论应用的目标 SDK 版本和清单标记值是什么，您都可以在使用分区存储的情况下测试应用，只需开启此变更即可。

> ★ **注意**：开启此变更以进行应用测试时，您还必须启用 DEFAULT_SCOPED_STORAGE（如果尚未将其启用）。

如需详细了解 Android 11 中分区存储的变更，请转到关于 Android 11 中 Android 存储变更的页面，请参阅分区存储部分。

GET_DATA_CONNECTION_STATE_R_VERSION

变更 ID：148535736

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

检查 `PreciseDataConnectionState#getDataConnectionState` 的 SDK 版本。

GET_DATA_STATE_R_VERSION

变更 ID：148534348

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

检查 `getDataState()` 的 SDK 版本。

GET_PROVIDER_SECURITY_EXCEPTIONS

变更 ID：150935354

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11（API 级别 30）或更高版本为目标平台的应用，`getProvider(String)` 不再抛出任何安全异常。

GET_TARGET_SDK_VERSION_CODE_CHANGE

变更 ID：145147528

默认状态：对于以 Android 10（API 级别 29）或更高版本为目标平台的应用处于启用状态。

检查 `SmsManager.sendResolverResult()` 方法的 SDK 版本。

> ★ **注意**：此变更目前包含在兼容性框架中，但会影响先前 Android 版本的 SDK 版本检查。您可以放心地忽略它。

GWP_ASAN

变更 ID：135634846

默认状态：对于所有应用处于停用状态。

在应用中启用采样原生内存错误检测。

如需详细了解此变更，请参阅 GWP-ASan 指南。

HIDE_MAXTARGETSDK_P_HIDDEN_APIS

变更 ID：149997251

默认状态：对于以 Android 10（API 级别 29）或更高版本为目标平台的应用处于启用状态。

对于以 Android 10（API 级别 29）或更高版本为目标平台的应用，移除了对 Android 10（API 级别 29）的 max-target-p（greylist-max-p）列表中所有非 SDK 接口的访问权限。

HIDE_MAXTARGETSDK_Q_HIDDEN_APIS

变更 ID：149994052

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11（API 级别 30）或更高版本为目标平台的应用，移除了对 Android 11（API 级别 30）的 max-target-q（greylist-max-q）列表中所有非 SDK 接口的访问权限。

如需详细了解此变更，请参阅 Android 11 中现在被屏蔽的非 SDK 接口。

LISTEN_CODE_CHANGE

变更 ID：147600208

默认状态：对于以 Android 10（API 级别 29）或更高版本为目标平台的应用处于启用状态。

检查 `TelephonyManager.listen(PhoneStateListener, int)` 的 SDK 版本。

> ★ **注意**：此变更目前包含在兼容性框架中，但会影响先前 Android 版本的 SDK 版本检查。您可以放心地忽略它。

MISSING_APP_TAG

变更 ID：150776642

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的应用，如果应用的清单文件缺少 `application` 或 `instrumentation` 标记，系统现在会抛出错误。

NATIVE_HEAP_POINTER_TAGGING

变更 ID：135754954

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的应用，现在原生堆分配在最高有效字节中有一个非零标记。

如需了解详情，请参阅堆指针标记。

PHONE_STATE_LISTENER_LIMIT_CHANGE_ID

变更 ID：150880553

默认状态：对于所有应用处于启用状态。

对于以 Android 11 为目标平台的应用，现在对任何进程可通过 TelephonyManager.listen(PhoneStateListener, int) 注册的 PhoneStateListener 对象数量施加了限制。默认限制为 50，该值可能会因远程设备配置更新而发生变化。当违规进程试图注册过多的监听器时，TelephonyManager.listen(PhoneStateListener, int) 会抛出 IllegalStateException，从而强制执行此限制。

PREVENT_META_REFLECTION_BLACKLIST_ACCESS

变更 ID：142365358

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

以 Android 11 为目标平台的应用不能再使用额外的反射层访问受限制的非 SDK 接口。

PROCESS_CAPABILITY_CHANGE_ID

变更 ID：136274596

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的应用，现在可以使用 Context.BIND_INCLUDE_CAPABILITIES 标记将正在使用的功能从客户端进程传递到绑定服务。

REMOVE_ANDROID_TEST_BASE

变更 ID：133396946

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的应用，已将 android.test.base 库移除，前提是应用不依赖于 android.test.runner（因为它依赖于 android.test.base 库中的类）。

REQUEST_ACCESSIBILITY_BUTTON_CHANGE

变更 ID：136293963

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

对于以 Android 11 为目标平台的无障碍服务，现在必须在无障碍服务元数据文件中指定 FLAG_REQUEST_ACCESSIBILITY_BUTTON 标记。否则，该标记会被忽略。

如需详细了解此变更，请参阅在元数据文件中声明“无障碍”按钮使用情况。

RESOURCES_ARSC_COMPRESSED

变更 ID：132742131

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

如果以 Android 11（API 级别 30）为目标平台的应用包含压缩的 resources.arsc 文件或者如果此文件未按 4 字节边界对齐，应用将无法安装。

如需详细了解此变更，请参阅压缩的资源文件。

RESTRICT_STORAGE_ACCESS_FRAMEWORK

变更 ID：141600225

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

如果您的应用以 Android 11 为目标平台并使用存储访问框架 (SAF)，您就无法再使用 ACTION_OPEN_DOCUMENT 和 ACTION_OPEN_DOCUMENT_TREE intent 操作访问某些目录。如需详细了解这些变更，请转到介绍 Android 11 中与存储相关的隐私权更新的页面，请参阅文档访问限制部分。

SELINUX_LATEST_CHANGES

变更 ID：143539591

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

此变更会控制应用对 untrusted_app_R-targetSDk SELinux 域的访问。这是兼容性框架的其中一项基础变更，可让应用在不更改其 targetSDKVersion 的情况下，切换受 targetSdkVersion 控制的其他变更的状态。因此，对于以 Android 11 为目标平台的应用，您不应停用此变更，否则应用将无法运行。

此变更不会影响使用共享用户 ID 的应用。

THROW_SECURITY_EXCEPTIONS

变更 ID：147340954

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

在 Android 11 之前，SecurityException 只会因权限错误而被 setEnabled API 抛出。在 Android 11 中，情况不再如此，SecurityException 可以因多种原因而被抛出，并且这些原因不会为调用方所知晓。

为了维持现有的 API 行为，如果不是以 Android 11 为目标平台的应用发生原来的权限失败或操作方强制执行失败，该异常会被强制转换为存在于 Android 11 之前的源代码中的 IllegalStateException。

USE_SET_LOCATION_ENABLED

变更 ID：117835097

默认状态：对于以 Android 11（API 级别 30）或更高版本为目标平台的应用处于启用状态。

以 Android 11 为目标平台的管理应用不能再使用 DevicePolicyManager.setSecureSetting(ComponentName, String, String) 更改已弃用的 Settings.Secure.LOCATION_MODE 设置，而应改用 DevicePolicyManager.setLocationEnabled(ComponentName, boolean)。

返回主目录

第四章

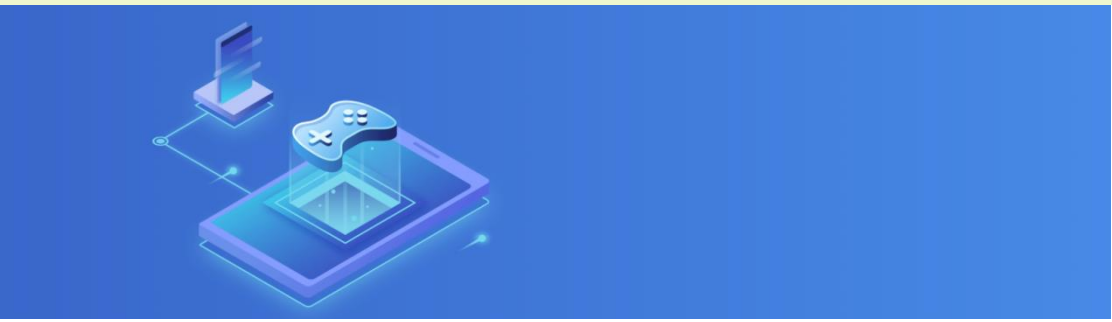
开发者案例

目录

返回主目录

卓盟科技：

Android 11 新功能让开发效率和质量大幅提升



在游戏变得日益复杂的今天，一根加载进度条不再仅仅只是开启冒险的倒计时，更是连接玩家与开发者的信使。

玩家们总是希望游戏能马上加载完毕，这意味着“加载”这件事情本身也有轻重缓急： 游戏开始几分钟内会被用到的资源需要打包进 APK，其他的内容则等玩家启动游戏后在后台下载。

运营总是希望游戏里有数不清的新花样，这意味着“变化”这件事情本身成为了“常态”：不同的节日需要不同的开屏和主题风格，甚至连游戏的观感都要让玩家觉得“啊，原来今天也是重要的一天呢”。

上海卓盟信息科技有限公司的乐变游戏分包服务，通过在游戏加载及运行时动态下载并插入新资源，来帮助游戏开发者满足玩家的需求。

网易的《流星群侠传》、龙渊的《多多自走棋》、紫龙的《梦幻模拟战》、游族的《少年三国志 2》……等诸多厂商的游戏，带给玩家们的体验也各不相同。但它们有一个共同点：都使用卓盟科技的乐变游戏分包服务来动态加载资源。

甚至就连卓盟科技自己，也需要借助动态加载资源来为自家 SDK 的用户们——也就是游戏厂商——提供更好的开发体验。比如让 SDK 的集成尽量简单，以及在需要调整 SDK 内部资源的时候提供快捷修复，无需开发者们更新 SDK。

在 Android 11 提供 ResourcesLoader 接口之前，卓盟科技打造动态资源加载的过程并不轻松。

旧版解决方案

当卓盟科技开始构建其产品时，Android 并没有公开接口支持动态资源加载的用例。团队尽了最大的努力，最终使用了非公开接口添加外部资源。虽然这个实现满足了技术需求，但其实非常脆弱——它依赖于非公开接口，这些接口的兼容性保证远远低于官方公开 SDK，并且随时可能在没有提前通知的情况下被更改或删除。

卓盟科技发现，随着每次 Android 新版本的发布，兼容性问题会意外出现。这些都需要进行额外的测试和开发，以确保产品的稳定性。经过多次迭代团队一共花费了 6 个工程师*月和

大量的代码将其解决方案稳定下来，同时理解它可能还会在下一个 Android 版本中再出问题。随着 Android 严格限制非公开接口以提升应用的稳定性和兼容性，卓盟科技需要移除对那些非公开接口的依赖。

可持续的解决方案

随着 Android 系统团队越来越专注于帮助应用迁移到公开接口的工作，卓盟科技看到了彻底解决兼容性问题的曙光。他们联系了 Android 团队来提供反馈，说明他们的具体用例和对公开接口的详细需求。

在双方的多次沟通和持续协作下，Android 11 中首次发布了支持动态资源加载的公开接口。卓盟科技已经迁移至新的 ResourcesLoader 接口，并获得了生产率和产品质量的大幅提升。卓盟科技认为 ResourcesLoader 接口具有以下优势：

- **使用简单。**开发团队两天内便完成了从现有方案至新方案的切换及测试。
- **性能无损。**和直接将资源内置在包内相比并无降低（某些场景下甚至有提高，因为包内的部分资源是压缩的，而通过 ResourcesLoader 添加进去的资源是非压缩的）。
- **开发高效。**以前的方案需要高级工程师先去了解 AssetManager 的原理，找到对应的私有接口及其在各个系统版本上的实现细节；同时需要掌握 zip 文件结构等和 Android 开发无关的技术细节。而使用新的公开接口后，一个能看懂文档的初级工程师即可轻松应对。

- **维护简单。**之前的方案为了兼容各种情况，累计代码行数超过 1,000 行，而新的方案代码仅几行即可！
- **向前兼容。**通过使用 Android 官方支持的公开接口，开发者的解决方案将在未来的 Android 平台上具有更好的兼容性。

```
String sdkroot = getApplicationInfo().dataDir + "/lebian";ResourcesLoader rl = new ResourcesLoader();rl.addProvider(ResourcesProvider.loadFromDirectory(sdkroot, null));Resources res = getResources();res.addLoaders(rl);final AssetManager assetManager = res.getAssets();
```

△ 使用 ResourcesLoader API 之后，核心代码只需几行

性能提升

卓盟科技以 16,028 个文件（总大小 1.47GB，压缩后为 1.36GB）为例，使用四种加载方案进行测试：

- 直接从 APK 中读取
- 使用某些私有接口加载目录后读取
- 使用 ResourcesLoader 接口加载 APK 后读取
- 使用 ResourcesLoader 接口加载目录后读取

前三种方案都对资源文件进行了压缩，平均加载时间都在 19 秒左右；而方案四通过 ResourcesLoader 加载目录后直接读取未压缩的资源，平均加载时间仅为 3 秒左右，性能提高了 6 倍！

卓盟科技首席执行官兼产品负责人黄杲总结了这套新接口的作用：“新的 ResourcesLoader 接口大大降低了开发和维护成本，使我们能够更加专注于产品和业务创新。”

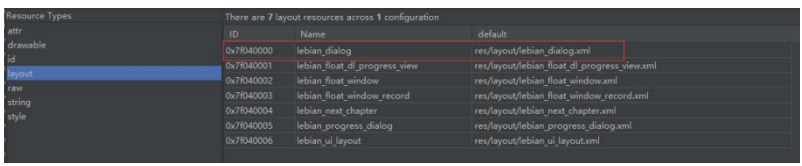
ResourceLoader 使用注意事项

使用 getIdentifier 时需要注意如下两点：

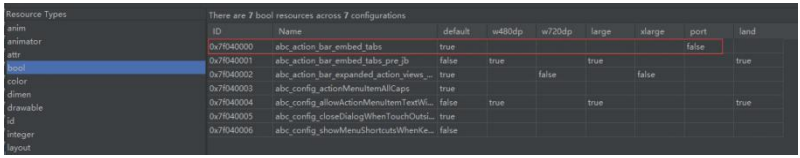
1. 通过 ResourcesProvider.loadFromApk(fd) 添加新的资源时，补丁 apk 资源的包名和宿主应相同，否则 getIdentifier 无法获取补丁的资源的 id；
2. 对于宿主中已经存在的资源，补丁资源名称和 id 应和宿主保持一致；对于新增的资源，不应与宿主中的 id 重复；

创建单独的 Resources 对象去加载补丁并访问其中的资源时，需注意如果补丁的资源的配置和宿主不同，则根据 id 去获取资源时，可能会拿到的是宿主的而不是补丁里的。

比如，补丁资源 plugin.apk 内有个资源 layout 资源只有 default 配置，id 是 0x7f040000，如下图：



宿主有个 bool 资源，id 也是 0x7f040000，但是有个竖屏的配置值为 false，如下图：



则在竖屏模式下，通过 res.getLayout(0x7f040000);时会报错 java.lang.RuntimeException: android.content.res.Resources$NotFoundExceptio: Resource ID #0x7f040000 type #0x12 is not valid

所以，如果您的使用场景不是用于资源修复，而是动态加载一个文件并访问其中的资源，比较保险的做法是不要和宿主中的 id 有重复。

共执画笔，共绘良图

“依靠 Android 平台，我们开发了一些有价值的产品和服务，这些产品也支撑着我们持续在 Android 平台投入更多的资源去开发更多创新的产品。”卓盟科技表示，“希望能有更多的机会参与到 Android 生态的建设中，贡献我们的绵薄之力，让 Android 对消费者更易用，对开发者更轻松。”

卓盟科技致力于提升其解决方案的长期兼容性。迁移至 ResourcesLoader 公开接口提升了稳定性和运行性能、简化了代码复杂性、降低了未来 Android 系统上的兼容性风险。更为重要的是，由于 ResourcesLoader 是 Android 11 中的公开接口，整个 Android 开发者社区都可以利用它来获得收益。

△ *卓盟科技团队*

Android 团队一如既往地重视开发者的反馈和创造力。动态资源加载接口的诞生，与开发者的支持与合作密不可分。我们也期待更多的开发者和卓盟科技一样，在 Android 生态中挥洒创意、创造价值、收获成功，同时也帮助 Android 成为开发者和用户们的绝佳平台！

更多精彩的开发者案例，欢迎访问“谷歌开发者”公众号“#Google 开发者故事”话题。也欢迎您和广大开发者分享您及您的团队在进行 Android 11 开发、迁移等时的实践经验、心得和故事！

返回主目录

第五章

常见问题

返回主目录

常见问题：

存储

Q: Android 11 的分区存储是强制的吗？如果 targetSdkVersion 低于 Android 10，运行在 Android 11 的手机上，分区存储特性还生效吗？

A: 当应用的 targetSdkVersion 升级到 Android 11 时，分区存储特性会强制生效。但如果应用 targetSdkVersion 未升级到 Android 11，运行在 Android 11 系统上时，分区存储不会强制生效。但根据 Google Play 的政策，在每一个 Android 大版本发布之后的次年 8 月，所有新发布的应用 targetSdkVersion 都需要升级至该版本或更高版本，且在版本发布的次年 11 月，所有应用 targetSdkVersion 都需要升级至该版本或更高版本。

Q: 清理工具类应用如何帮助用户清理应用专属目录中的数据？

A: MANAGE_EXTERNAL_STORAGE 权限一般适用于清理、文件管理、备份或恢复类型的应用，并且该权限会由 Google Play 来控制保护权限不会被滥用。清理类应用可以访问所有的外部存储，但同样也无法访问其他应用的专属目录。在分区存储中，应用的专属目录可以理解为和内部存储是等同的，在 Android 11 中也是不可以去访问的。如果清理类应用可以访问其他应用的专属目录，那么为了保护自己的数据，应用还是会选择将数据存放至内部存储中，这

就和分区存储的出发点有异议了，所以也可以认为应用的专属目录就相当于是 "内部存储"。

Q: Android 11 以后，文件管理器或清理大师之类的第三方应用是不是就没有机会访问其它应用专属区域产生的文件了？

A: 是的，如果第三方的文件管理应用还有机会去访问其他应用产生的专属目录里的文件，那么这些应用就可以进一步选择将应用文件放在内部存储中，这样对于外部存储来说并不是一个很好的应用规范。

Q: 分区存储允许应用通过 File API 使用文件路径访问文件吗？

A: 我们意识到某些应用会通过代码或程序库直接访问媒体文件路径。因此，在 Android 11 上，拥有可读取外部存储权限的应用，均可在分区存储环境中通过文件路径访问文件。在 Android 10 的设备上，除非在 manifest 中通过主动声明 requestLegacy ExternalStorage 属性来选择停用分区存储，否则上述方法是无效的。

为了确保不同 Android 版本间的连续性，如果您应用的目标版本是 Android 10 或者是更高版本，您应该选择不启用。更多详细信息，请参阅 Android 存储方案的最佳实践上篇、下篇。

Q: 与媒体存储 API 相比，文件路径访问的性能表现如何？

A: 性能表现非常依赖具体应用场景。对于像视频播放这样的拥有顺序读取的操作，文件路径访问的性能表现与媒体存储相差无几。但

是在随机读写的情境下，采取文件路径的方法最多可慢一倍。为了最快、最稳定的读写，我们推荐您使用 Media Store API。

Q: 我的应用需要广泛地访问共享存储，存储访问框架是我唯一的选择吗？

A: 存储访问框架（简称 "SAF"）用于用户授予对目录和文件的访问权限，但是需要您注意的是，SAF 对某些目录的授权仍存在限制，例如根目录和 Android/data 目录。虽说大多数应用在存储访问时都可以通过我们最佳实践的方式去实现，例如使用 SAF 或媒体存储 API，但在某些应用场景下可能会需要更广泛地访问共享存储，亦或是无法通过最佳实践来有效地访问。针对上述情况，我们增加了 MANAGE_EXTERNAL_STORAGE 权限，允许程序访问外部存储上的所有文件（除了 Android/data 和 Android/obb 目录）。我们在 7 月发布了一个 Google Play 政策更新，提到了关于存储的相关内容，请点击这篇微信文章查看。

Q: 哪些类别的应用应该申请 MANAGE_EXTERNAL_STORAGE 权限？

A: MANAGE_EXTERNAL_STORAGE 权限适用于核心应用场景需要广泛地访问设备上的文件的情况，但使用分区存储的最佳实践无法高效地实现此功能的那些应用。当然，列出所有可能的应用场景是不切实际的，但其包括了文件管理器、备份和还原、反病毒程序或生产力文件编辑器等使用场景。

Q: 使用 Storage Access Framework (存储访问框架)，是否需要 Google Play 的政策批准？

A: Storage Access Framework（存储访问框架，简称 SAF）从 Android 4.4 开始就已经存在。通过 SAF 访问文件时，会让用户参与文件选择，从而使用户可以更好地控制文件的访问。Google Play 上没有与之相关的政策。

Q: 与 Android 10 相比，在 Android 11 上使用 SAF 会有其他限制吗？

A: 目标版本为 Android 11 (API 级别为 30) 并使用 SAF 的应用，将不会被授予某些目录访问权限，例如 SD 卡上的根目录和下载目录。无论是哪个目标 SDK，都无法在 Android 11 上通过存储访问框架访问 Android/data 和 Android/obb 目录。访问官方文档了解关于这些限制和测试相关行为的方法。

Q: 应用该如何测试分区存储的变化？

A: 通过这些兼容性标志，应用可以测试与直接文件路径访问或媒体存储 API 相关的分区存储行为。还有另一个兼容性标志，也可用来测试使用存储访问框架访问某些路径时的限制。

Q: 分区存储中的应用，是否仅限于将文件写入其应用 data 目录上？

A: 在分区存储中，应用可以将媒体项添加到媒体存储集合。媒体存储会根据文件类型，将文件放置于组织有序的文件夹中，例如 DCIM、Movies 和 Download 等。对于所有此类文件，应用可以继续通过文件 API 来访问。得益于系统为每个媒体存储文件赋予了应用属性，应用不需要有存储权限也可以读写到它们最初提供给媒体存储的文件。

Q: Data Column 弃用之后，有没有对此功能的其他使用建议？

A: 在 Android 10 上，位于分区存储环境中的应用无法通过文件路径访问文件。为了与这一设计保持一致，我们随后废弃了 DATA column。根据大家的反馈，即需要使用已有的 native 代码或程序库，Android 11 现已支持在分区存储中的应用访问文件路径的功能。相应地，DATA Column 实际上在某些情况下其实是有用的。为了在媒体存储中插入和更新，使用分区存储的应用应使用 DISPLAY_NAME 和 RELATIVE_PATH Column，它们不再需要使用 DATA Column 来完成此功能。当读取磁盘中文件的媒体存储实例时，DATA Column 将具备有效的文件路径，该路径可被文件 API 或 NDK 文件程序库使用。但应用要准备处理任何关于此类操作带来的 I/O 错误，而且不应该假设文件始终是可用的。

Q: 对于选择退出分区存储的应用，它们何时开始必须兼容分区存储？

A: 在运行 Android 11 或更高版本的设备上。当目标版本被设置为 Android 11 或更高版本时，应用便会被放入到分区存储中。

Q: 建议使用什么方法来迁移分区存储之外的数据？

A: preserveLegacyExternalStorage 标记允许应用在升级系统时保留原有存储权限，即使是升级至 Android 11。需要注意的是，在 Android 11 上这个标记对新安装的应用起不到任何作用。将目标版本设为 Android 11 之前，请修改代码以适配分区存储。请参阅文末相关阅读 Android 存储方案的最佳实践上下篇，来获取数据迁移最佳实践的相关信息吧。

Q: 针对某些软件包安装程序（例如应用商店）需要访问 Android/obb 目录，是否有任何例外情况?

A: 持有 REQUEST_INSTALL_PACKAGES 权限的应用可以访问其他应用的 Android/obb 目录。请注意，此权限享有签名级别的保护。

权限

Q: 对于厂商自定义权限，没有采用 Google 的权限设计方式，导致应用开发各种兼容问题，是否考虑让厂商统一？比如浮窗权限(甚至影响 Toast 的使用)，应用列表获取权限，各种 Google 没有定义的 sensor 权限。

A: 如果各位开发者在开发过程中，发现应用在不同的手机上面的表现不一致，特别是如果认为某个手机的行为逻辑与 CDD 相悖，可以向我们反馈。通常 Android SDK 在所有 Android 手机上的表现应该都是一致的，但我们也知道某些厂商会增加一些新的功能，比如某些手机有一个电量优化的功能，对手机通知功能有影响，可能一定程度上影响用户体验及应用功能。

如果有一些功能只在部分手机厂商上存在，因为 Android 是开源的，厂商可以自行增加新的功能。对于这部分也希望各位开发者向我们反馈，让我们了解到一些功能对开发者和用户比较友好，希望可以增加到 AOSP 中。我们也可以和厂商反馈，连同厂商、开发者和用户一起把整个 Android 做的更好。

Q: shouldShowRequestPermissionRationale 什么时候是 true？依据是什么？被 denied 过一次吗？

A: 因为这个是系统级 API，所以只需要去调用并且按照返回值来做合适的操作就可以了。原则上来说，如果用户拒绝过一次，且拒绝时未选择 "don't ask me again" 选项，那么下一次返回值应该就是 true。如果想要了解更详细的实现细节，可查看 AOSP 中对应的源代码。对于应用开发者来说主要还是根据其返回值来判断即可。 可参阅相关文档。

Q: 对于单独进程 (单独开了个服务，指定进程名，为后台进程)，Android 11 对定位是否有影响？

A: 如果进程是后台进程，应用需要有后台定位权限才可获取位置信息。针对一些特殊情况会有针对处理，比如应用在后台但开启前台服务，通过一个持续性的通知让用户感知其在后台运行，在这种情况下我们会认为该应用是前台应用，那么应用有前台定位权限就可获取位置信息。

Q: 访问后的回调是否会精确到具体是哪个接口涉及到某项敏感信息？比如 requestLocationUpdate 涉及位置信息。

A: 设置数据访问操作回调 API 还处于 Developer Perview 阶段，后续会根据实际需求不断改进。类似的适用场景比如有些时候我们需要去了解用户的行为可能和某些权限有关，并且涉及权限和应用代码要求是否是一致的。如果已经知道具体的操作是通过哪些代码实现的，那就不需要使用这个 API。如果您不知道是通过哪些代码

实现，或者是否是第三方库运行结果，那通过这个 API 会有很大帮助。具体还是要参考实际用例。

Q: Android 11 会禁用应用修改系统的位置吗？或者检测应用是否使用虚拟定位？

A: 如果修改系统位置可能需要 Root 权限，这样就不是我们常规考量的用户体验了。关于检测是否使用虚拟位置，一些开发者的做法是通过检测当前设备上有没有装一些专门用于修改位置的应用来实现的，如果在 Android 11 中需要实现，需要考虑应用可见性，在 mainfest 文件中列明需要检测的应用包名即可。

Q: 一次性权限是要一直申请么？有没有白名单机制，比如我是相机应用，如果一直申请相机权限，可能会有一些体验问题。

A: 一次性权限是由用户来授予的，应用是不能显式申请一次性权限的。应用执行的是告知用户要申请权限，然后用户选择给予的是一次性权限还是长期的权限，这对于应用来说是透明的。对于应用开发者，我们建议按照实践指南来开发，在每次需要使用权限时，应该检查是否获得相应权限，如果没有的话按照实践指南去申请对应权限。当用户授予对应权限后就可以继续运行；当用户没有授予相应权限，也可以提示用户解释为什么需要权限，让用户了解到权限申请的必要性。可参阅一次性权限相关文档。

CameraX

Q: CameraX 是否会和更多的厂商合作，提供定制化的功能？

A: 我们已推出了一个 Jetpack 支持库 CameraX，旨在简化相机应用的开发工作。目前我们针对 CameraX 有合作的厂商包括三星、OPPO、小米、LG 等，还有一些其他厂商自己也在跟进。更多的还是取决于厂商自己对于手机功能开发的意愿，并且由于 CameraX 是一个开源的项目，我们不需要直接跟厂商去合作。

新的屏幕类型

Q: Android 11 对于折叠屏的支持有改进吗？

A: 在 Android 11 中新增了一些针对折叠屏设备状态的 API，比如在第 2 个开发者预览版中新增了 API 来检测设备铰链的开合角度，这样应用就可以根据铰链的开合角度和位置显示不同内容。

隐私/安全

Q: Android 系统关于防破解如何从底层提供更好的支持？

A: 国外开发者只需将应用上传至 Google Play 应用商店即可通过 Google Play 的安全防护机制有效的保护游戏和玩家利益、减少游

戏被篡改和盗版的问题。如果有应用被破解或上传至 Play 应用商店，原开发者可以要求 Play 查明后进行下架处理。

而国内生态目前是比较碎片化，有很多发布渠道，所以防破解是个比较重要的需求。在了解到这个情况之后，我们也和很多的加固厂商建立了密切的联系。从 Android O 开始我们就一直在支持一些厂商的合理需求，比如 Android 10 中新增了一个新的公开接口，AppComponentFactory.instantiateClassLoader()，在应用被加载运行其他代码之前就创建并设置一个自定义的 ClassLoader，满足加固和热修复方案的需求。在 Android 11 中，我们又增加了 ResourcesLoader API，能够让加固和热修复方案通过系统支持的接口来做自定义的资源加载。我们也会持续和加固厂商合作提升加固，尽量少的使用私有 API，以全面的获得系统级别的支持。对于应用开发者来说，在选择了这些加固方案之后，也可以更好的保证应用对未来版本的 Android 系统兼容性。

Q: 在 <queries> 里面的 intent action 写 android.intent.action.Main 是不是就相当于可以查询所有 App 是否已安装？因为几乎所有 App 都会申明这个 intent action。

A: 之所以添加应用包可见性的改动，是为了保障用户隐私，不希望应用可以随意查询手机上所有应用。目前对于哪一些 action 可以查询是没有限制的，但相信在最终版本中是不允许对 android.intent.action.Main 进行查询的，无法获取结果。

Q: 灰名单的限制具体是哪些？是其他 jar 包访问不到？

A: 灰名单和其他 jar 包没有关系。无论在任何渠道，目前调用浅灰

名单没有问题，但无法保证在未来版本浅灰名单中的非 SDK 接口是否会移至黑名单，所以我们建议浅灰名单中的非 SDK 接口尽量减少调用。如果非 SDK 接口在深灰名单，就表明该非 SDK 接口是根据 targetSDKversion 区分，不同版本对于是否可以调用限制不同。这个限制和 jar 包无法访问是没有关系的，不管是从哪里调用这个接口。可参阅非 SDK 接口限制相关文档。

Q: 国外可以通过 Google Play policy 去控制 targetSdk Version。国内环境，APK 的 targetSdkVersion 可以不升级，也可以安装使用，继续访问用户隐私数据 Android 新系统，对这个方向有没有什么想法 ？

A: 其实我们也能看到国内的生态也对隐私和安全性越来越重视，之前也有看到工信部联合主流国内应用商店，对 targetSdkVersion 有了升级要求，虽然可能会相比我们对这个要求慢一些，但至少也是在不断升级。另外对于 targetSdkVersion 比较老的应用，从 Android 10 开始也会有一些系统层面的限制，比如 targetSdkVersion 小于 17 的应用，在安装和每次运行时都会给到用户提示警告。这些措施也都是希望联合整个 Android 生态中所有的参与者一起推进 targetSdkVersion 的升级，希望通过所有应用都升级至高版本的 targetSdkVersion 来让用户获得更好的安全性和隐私方面的体验。

应用包可见性

Q: 应用包名可见性会不会影响 deeplink、applink 等功能?

A: 应用包名可见性的改动不会影响 deeplink 和 applink，因为底层系统会有一些改动，所以对于应用来说继续使用 deeplink 和 applink 理论上是不会有改动影响的。如果您要启动新的服务或启动过其他的应用，如果您的应用不可以看到其他的应用，是无法启动其他应用的组件。

API

Q: Android 10 或者 Android 11 中使用了黑名单或者灰名单的 API 后，会被 Google Play 应用商店拒绝吗?

A: 如果应用使用了黑名单中的接口，运行时可能会有异常从而导致应用无法正常使用，那么 Google Play 是会拒绝上架的。私有名单的限制是 Android 系统层面执行的，我们做这个限制的目的并不是为了限制开发者，当开发者应用遇到问题时可以考虑是否必须要使用这个接口，或者也可以向我们反馈告知合理需求，希望开放公开的 SDK 接口来满足相应需求。

SDK

Q: 对于 targetSdkVersion 非 Android 11 的应用会有什么影响吗?

A: 我们在每一次更新新的版本的时候会考虑尽量减少对于应用的影响。从这个角度来看，我们会尽量把行为变更放在 targetSdkVersion 升级之后。如果应用还没有升级到最新版本的 targetSdkVersion，就不会受到行为变更的影响。如果现有您的应用已经遵循我们的实践指南，受到变更的影响就会很小。但因为在 Android 11 中我们对系统底层也做了一些改动，比如权限管理、一次性权限还有分区存储的一些变更，我们也希望大家可以在 Android 11 模拟器或真机中调试自己的应用，以确保没有问题。可参阅 Android 11 行为变更指南。

Q: 是否能再解释一下 targetSdkVersion 的工作机制，比如运行在 Android 11 上的 App，Android 会根据各 App 的 targetSdkVersion = 30 / 29 / 28 来执行不同的代码吗?

A: 我们在每次发布新版本的 Android 时，比如 Android 11，改动会分为两类。第一类就是长期性的改动，不论 targetSdkVersion 版本，只要在系统中运行时改动就会长期执行。第二类就是根据 targetSdkVersion 来决定具体行为。如果改动是按照 targetSdkVersion 来决定，有一些改动只适用于 targetdkVersion 30 以上，那么您的应用 targetSdkVersion 是 29 或者以下的话，这些改动是不会影响您的应用行为的。只有升级到 targetSdkVersion 30 或以上这些改动才会生效。可参阅相关文档。

其他

Q: 关于 Android 虚拟机近期有什么更新?

A: 我们已发布了关于虚拟机的相关更新介绍，目前最新版本的虚拟机支持直接运行 ARM 应用，无需再构建 x86 版本，可以直接使用 ARM 版本。虚拟机可以高效地直接将 ARM 指令转换成 x86 指令，直接运行 ARM 应用也可以获得很好的性能。欢迎大家体验尝试。

Q: Android 是否考虑采用方法传递回调参数的形式解决回调，现在这种 Activity 的回调形式用起来很不方便。

A: 在 Java 当中，如果大家惯用 Java 的 Callback，用起来可能会觉得语法比较复杂，其实在其他语言里都有支持 Lambda Expression 和 Functional Programming 的类似概念。如果大家还没有使用 Kotlin 的话，我们强烈建议大家可以去尝试一下，因为在 Kotlin 里对 Lambda 表达式算是 "一等公民" 的支持。如果大家情况允许的话可以尝试去选用 Kotlin。

Q: 画中画的窗口大小现在能支持自定义了吗?

A: 目前还不可以，可以通过 setAspectRatio() 来设置宽高的比例，但是不可以指定一个具体的尺寸。画中画的窗口大小应该由用户去决定。

Q: <queries> 支不支持代码动态设置?

A: 目前来说 <queries> 标记只支持在 manifest 里面定义，不能在应用运行时动态设计。

Q: 一般 App 可能会有渠道包，例如 com.example.xiaomi com.example.huawei，在 manifest <queries> 是否能写 com.example.*?

A: 目前在 manifest 中使用 getPackageInfo 或者 getInstalledPackages 去查询应用是否安装，必须要使用完整的应用包名才可以。

Q: 输入法动画有 Demo 参考吗？低版本如 Android 10 有办法使用吗?

A: 我们 GitHub 中我们提供了相应示例，通过参考示例代码可以有更完整的了解。目前只有在 Android 11 中可以使用这个最新的 API，在低版本中无法使用。我们也会去评估是否可以支持在低版本的 Android 系统中使用。

Q: ApplicationExitInfo 的崩溃退出信息有多详细？是在崩溃后下一次启动 app 才能获取?

A: 当应用崩溃之后，相关信息会存储在缓冲区，在应用即时可以存取。一般来说，当应用崩溃之后，用户会尝试再次打开应用，所以可以在下一次打开的时候将崩溃退出信息汇报至后台，开发者可以去查看分析具体是什么原因导致的。

Q: OBB、AAB 的功能有重叠，OBB 的初始化下载能否有保障？或者对于 OBB 的最佳实践是哪些场景?

A: 其实 OBB 和 App Bundle 之间本身是没有重叠的，OBB 是为了带有很大资源包的游戏所单独设计的，Play 允许为每个游戏添加最多两个 OBB 文件，每个的上限是 2GB，所以最大可以包含 4GB 的资源包。而 App Bundle 是有比较严格的下载大小限制的，无法实现下载 4GB 的资源包，可以简单的理解为 OBB 是为游戏打造的，而 App Bundle 是为其他应用打造的。我们已专门为游戏应用推出了一个跟 App Bundle 类似的产品，名为 Play Asset Delivery，让游戏也可以被拆分成多模块按照 App Bundle 的方式去分发。可参阅“Google Play 优化高质量游戏交付”。

Q: Android 11 对于无障碍模式是否有限制或功能增强？

A: Android 在最近几年的版本中一直对无障碍模式、accessibility 都有越来越好的支持。在 Android 11 中也不会有更多的限制，并且还会有一些功能上的增强。请大家继续关注 Android 11 进展，我们在未来的开发者预览或者 Beta 版本中可能就会有一些关于无障碍模式的新功能公布。

返回主目录

第六章

相关资源

返回主目录

相关资源：

Android Developers 中文网站

https://developer.android.google.cn/

谷歌开发者微信公众号：Google_Developers

腾讯视频 Google Developers 频道

http://v.qq.com/vplus/78ffb35ddcf6577ad6e00555d6a4fbaa

Bilibili 视频谷歌中国官方帐号

https://space.bilibili.com/64169458

欢迎您提意见！

我们一直致力于不断改善我们的产品和服务，以帮助您更好的了解、使用我们的产品，拓展您和应用业务。

欢迎您关注谷歌开发者公众号，在留言板进行留言，与我们分享您对本手册的看法和意见建议，让我们了解您想在未来版本中看到那些内容。

返回主目录