安卓13来了，快！扶起我来！

江江安卓 郭霖 2022-07-19 08:00 发表于江苏

/ 今日科技快讯 /

近日，社交媒体Facebook母公司Meta旗下照片分享应用Instagram发生宕机事件，成千上万名用户受到影响。据宕机追踪网站Downdetector称，美国东部时间周四下午5点左右有近2.4万名Instagram用户受到宕机事件的影响。

/ 作者简介 /

本篇文章来自Zhujiang的投稿，文章主要分享了安卓13的新功能及特性，相信会对大家有所帮助！同时也感谢作者贡献的精彩文章。

Zhujiang的博客地址：

> https://juejin.cn/user/3913917127985240

/ 前言 /

一年一年过的太快了，还记得两年前写了 Android 11(R) 的适配文章，这一转眼都13(T)了，这样算下去几年后26个字母就用完了，到时候也不知道 Google 会如何进行命名😂。

下面咱们来看看 Android 13 都有哪些更新，并来看看开发者应该如何进行适配吧！

/ 隐私及权限相关 /

通知的运行时权限

在之前版本中我们应用如果需要弹通知的话只需要通过 NotificationManager 即可直接进行弹出，不需要任何权限，之前我一直觉得 Google 官方这一点做的不好，通知这么重要竟然不需要用户同意就可以直接弹出，当然你可以在设置中进行手动关闭，但这对于大多数人来说比较困难。然后在 Android 13（T-33）中终于引入了新的运行时权限——通知权限：POST_NOTIFICATIONS。

但是如果用户拒绝通知权限，他们仍会在前台服务 (FGS) 任务管理器中看到与这些前台服务相关的通知，但不会在抽屉式通知栏中看到这些通知。

这个更改对许多应用都有关系，只要你的应用会弹通知，那么如果要适配 Android 13 的话就都需要进行适配，当然适配方法很简单，再按照别的运行时权限适配下新的通知权限即可。

检查您的应用能否发送通知

如果想要确认用户是否已启用通知，可以调用 NotificationManager.areNotificationsEnabled() 来进行判断。

附近 Wi-Fi 设备的新运行时权限

在以前的 Android 版本中，需要 ACCESS_FINE_LOCATION 权限，应用才能完成与热点相关的多个常见 Wi-Fi 用例、Wi-Fi 直连、Wi-Fi RTT 等。

由于用户很难将位置信息权限与 Wi-Fi 功能相关联，因此 Android 13（T-33）在 NEARBY_DEVICES 权限组中引入了新的运行时权限，适用于管理设备与附近 Wi-Fi 接入点连接情况的应用。此权限 (NEARBY_WIFI_DEVICES) 可满足这些 Wi-Fi 用例。

只要应用不通过 Wi-Fi API 推导物理位置，那么在 Android 13 或更高版本为目标平台并使用 Wi-Fi API 的时候就可以请求 NEARBY_WIFI_DEVICES 而不是 ACCESS_FINE_LOCATION。

细化的媒体权限

如果要将应用升级为 Android 13 ，必须请求一个或多个新权限，Android 13 中将媒体权限细分为了图片、视频和音频文件，而不是之前的 READ_EXTERNAL_STORAGE 和 WRITE_EXTERNAL_STORAGE 权限。请求的权限集取决于应用需要访问的媒体类型，如下图所示：

媒体类型	请求权限
图片和照片	READ_MEDIA_IMAGES
视频	READ_MEDIA_VIDEO
音频文件	READ_MEDIA_AUDIO

注意：如果应用只需要访问图片、照片和视频，应该考虑使用照片选择器（下面会介绍），而不是声明 READ_MEDIA_IMAGES 和 READ_MEDIA_VIDEO 权限，还有，申请了最新的三个权限的话应用就无需再声明 WRITE_EXTERNAL_STORAGE 权限了。

下面来看下在 AndroidManifest.xml 中应该如何进行修改：

```
<manifest ...>
  <!-- Android 13 -->
  <uses-permission android:name="android.permission.READ_MEDIA_IMAGES" />
  <uses-permission android:name="android.permission.READ_MEDIA_AUDIO" />
  <uses-permission android:name="android.permission.READ_MEDIA_VIDEO" />

  <!-- Required to maintain app compatibility. -->
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
            android:maxSdkVersion="32" />
  <application ...>
    ...
  </application>
</manifest>
```

精确闹钟的新权限

如果升级到 Android 13 ，可以使用自动授予应用的 USE_EXACT_ALARM 权限。不过，一般是系统应用才可以使用，因为即将推出的 Google Play 政策会阻止应用使用 USE_EXACT_ALARM 权限，除非应用为日历或者时钟这样的系统应用（国内另说）。

如果应用设置了精确闹钟，但又不是系统日历或时钟的话，还是继续声明 SCHEDULE_EXACT_ALARM权限，并要为用户拒绝授予应用相应访问权限的情况做好准备。

开发者可降级权限

从 Android 13 开始，应用可以撤消先前由系统或用户授予的运行时权限。开发者可以：

- 撤消未使用的权限。
- 遵循权限最佳做法，从而提高用户信任度。可以向用户显示一个对话框，其中会显示应用主动撤消的权限。

如需撤消特定运行时权限，请将该权限的名称传入 revokeSelfPermissionOnKill()。如需同时撤消一组运行时权限，请将这组权限的名称传入 revokeSelfPermissionsOnKill() 。撤消是异步发生的，会终止与应用的 UID 相关联的所有进程。

为了使系统撤消权限，必须终止与应用关联的所有进程。当调用该 API 时，系统会确定何时可以安全终止这些进程。通常，系统会等待应用有较长时间在后台运行，而不是在前台运行时。

但如果为了立即撤消权限，那么就需要手动终止所有相关进程，但用户体验嘛，让产品自己取舍吧。

后台使用身体传感器新的权限

Android 13 中引入了“在使用时”访问身体传感器（例如心率、体温和血氧饱和度）的概念，如果要升级为 Android 13，并且在后台运行时需要访问身体传感器信息，那么除了现有的

BODY_SENSORS 权限外，还必须声明新的 BODY_SENSORS_BACKGROUND 权限。

如何申请运行时权限

想了下这块还是写的详细一些吧。Android 从 Android 6（M-23） 开始引入了运行时权限这个概念（所有权限列表），但是刚出来的时候编写比较费劲，于是乎就出现了一堆三方的权限库以简便申请权限的流程，这里就不一一进行列举了，相信大家也都知道或使用过，但现在官方对申请权限这块的代码进行了重写，使用起来并不比那些三方库复杂，甚至更加简单，下面来看下使用方法吧：

申请单个权限

```
val requestPermissionLauncher =
   registerForActivityResult(
      ActivityResultContracts.RequestPermission()
   ) { isGranted: Boolean ->
      if (isGranted) {
         // 同意
      } else {
         // 拒绝
      }
   }

when {
   ContextCompat.checkSelfPermission(
      this,
      Manifest.permission.CAMERA
   ) == PackageManager.PERMISSION_GRANTED -> {
      // 当前拥有这个权限
   }
   shouldShowRequestPermissionRationale(Manifest.permission.CAMERA) -> {
      // 告诉用户为啥要申请这个权限
   }
   else -> {
      // 申请权限
      requestPermissionLauncher.launch(
         Manifest.permission.CAMERA
      )
   }
}
```

代码比较容易理解，官方新封装的权限申请代码还是挺好的，无需咱们再自己处理 onRequestPermissionsResult 中的回调信息。

申请多个权限

```
val requestPermissionsLauncher =
   registerForActivityResult(
      ActivityResultContracts.RequestMultiplePermissions()
   ) {
      it.forEach { (name, success) ->
         if (success) {
            // 同意
         } else {
            // 拒绝
         }
      }
   }
when {
   // ...
   // 这块和上面基本一致
   else -> {
      // 申请多个权限，数组展示
      requestPermissionsLauncher.launch(
         arrayOf(Manifest.permission.CAMERA)
      )
   }
}
```

这块和上面申请单个权限的使用方法基本一致，只是将单个权限改为了多个权限。

剪贴板中隐藏敏感内容

从 Android 13 开始，将内容添加到剪贴板时，系统会显示标准视觉确认界面。新确认界面会执行以下操作：

- 确认内容已成功复制。
- 提供所复制内容的预览。

在 Android 12L（32）及更低版本中，用户经常不确定他们是否成功复制了内容或者复制了什么内容。

此功能可将应用在用户复制内容后显示的各种通知标准化，并让用户可以更好地控制剪贴板。

如果应用允许用户将敏感内容（例如密码或信用卡信息）复制到剪贴板，则必须在调用 ClipboardManager.setPrimaryClip() 之前向 ClipData 的 ClipDescription 添加一个标志。添加此标志可阻止敏感内容出现在内容预览中。

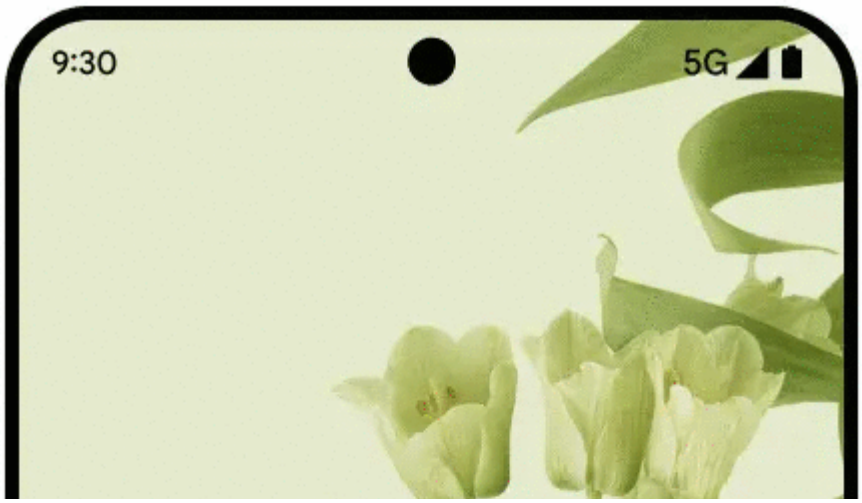
```
val clipboardManager = getSystemService(CLIPBOARD_SERVICE) as ClipboardManager

// When your app targets API level 33 or higher
clipData.apply {
   description.extras = PersistableBundle().apply {
      putBoolean(ClipDescription.EXTRA_IS_SENSITIVE, true)
   }
}

// If your app targets a lower API level
clipData.apply {
   description.extras = PersistableBundle().apply {
      putBoolean("android.content.extra.IS_SENSITIVE", true)
   }
}
```

/ 预测性返回手势 /

这个功能怎么说呢，苹果已经有的功能，由于现在 Android 13 还没有正式版，这个功能还不能进行测试，先来看看官方给的样子吧：



是不是和苹果的很像。。。。

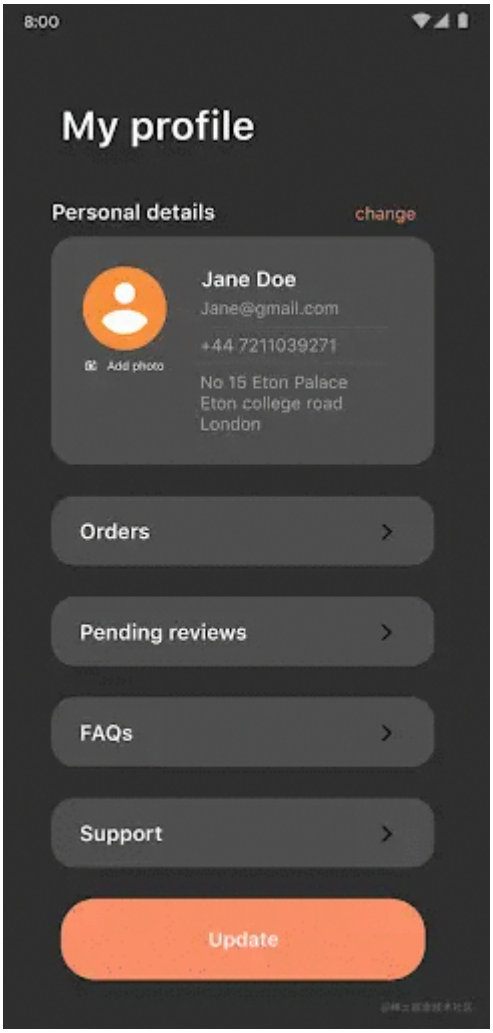
如需选择启用预测性返回手势，请在 AndroidManifest.xml 的 <application> 标记中将

```
<application
   ...
   android:enableOnBackInvokedCallback="true"
   ... >
...
</application>
```

enableOnBackInvokedCallback 默认值为 false，表示停用预测性返回手势。

/ 照片选择器 /

Android 13（T-33）支持新的照片选择器工具。此工具为用户提供了一种安全的内置媒体文件选择方式，让其无需向应用授予对整个媒体库的访问权限。



照片选择器提供了一个可浏览、可搜索的界面，其中按日期（从最近到最早）顺序向用户呈现其媒体库中的文件。可以指定用户只能看到照片或只能看到视频，并且默认情况下，允许的媒体选择量上限设置为 1。

定义分享限制

应用可以声明 android.provider.extra.PICK_IMAGES_MAX 的值，该值表示在向用户显示时照片选择器中显示的媒体文件数量上限。

需要注意的是：如果选择的上限为 1 张，照片选择器会以半屏模式打开。

选择单张照片或单个视频

先来看看如何选择单张照片吧：

```
val intent = Intent(MediaStore.ACTION_PICK_IMAGES)
// 用户可以选择一张照片或一个视频。
startActivityForResult(intent, PHOTO_PICKER_REQUEST_CODE)
```

选择多张照片或多个视频

如果应用的用例需要用户选择多张照片或多个视频，可以使用 EXTRA_PICK_IMAGES_MAX extra 指定照片选择器中应显示照片的数量上限，如以下代码段中所示：

```
// 最大选择数量
val maxNumPhotosAndVideos = 10
val intent = Intent(MediaStore.ACTION_PICK_IMAGES)
intent.putExtra(MediaStore.EXTRA_PICK_IMAGES_MAX, maxNumPhotosAndVideos)
startActivityForResult(intent, PHOTO_PICKER_MULTI_SELECT_REQUEST_CODE)
```

请注意，可指定为文件数量上限的最大数字存在平台限制。如需访问此限制，请调用 MediaStore.getPickImagesMaxLimit()。

处理照片选择器结果

照片选择器启动后，使用新的 ACTION_PICK_IMAGES intent 来处理结果。该选择器会返回一组 URI：

```
// 处理来自照片选择器的回调。
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent) {
    super.onActivityResult(requestCode, resultCode, data)
    if (resultCode != Activity.RESULT_OK) return
```

```
    when (requestCode) {
        REQUEST_PHOTO_PICKER_SINGLE_SELECT -> {
        // 获取单个选择的照片选择器响应
        val currentUri: Uri = data.data
        // 处理照片或视频的URI.
        return
    }
        REQUEST_PHOTO_PICKER_MULTI_SELECT -> {
        // Get photo picker response for multi select.
        var i = 0
        while (i < data.clipData!!.itemCount) {
            val uri = data.clipData.getItemAt[i]
            // 处理照片或视频的URI.
    }
        return
```

默认情况下，照片选择器会既显示照片又显示视频。咱们可以在 setType() 方法中设置 MIME 类型，以便按“仅显示照片”或“仅显示视频”进行过滤。来看看代码如何实现吧：

```
val intent = Intent(MediaStore.ACTION_PICK_IMAGES)
// 只显示视频
intent.type = "video/*"
startActivityForResult(intent, PHOTO_PICKER_VIDEO_SINGLE_SELECT_REQUEST_CODE)

// 只显示图片
// images only - intent.type = "images/*"
```

上面代码中将只显示视频或者图片的 type 都写了下，大家可以按需进行使用。

/ 应用内语言选择器 /

Android 13 在手机设置中新增了一个集中设置选项，用于设置各应用语言偏好设定。如果你的应用支持多种语言，官方强烈建议我们在应用的清单中声明 android:localeConfig 属性，这样用户就可以在同一位置像更改其他应用的语言设置一样更改应用的语言设置。

此外，当前使用自定义应用内语言选择器的应用应改用适用于各应用语言偏好设定功能的新 API。使用这些新 API 有助于确保用户无论是继续通过应用内语言选择器选择语言，还是通过手机设置选择语言，都能以其首选语言查看应用。当然，如果不支持多种语言的应用将不受这些变更的影响。

如何使用

1.创建一个名为 res/xml/locales_config.xml 的文件，并指定您的应用的语言，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<locale-config xmlns:android="http://schemas.android.com/apk/res/android">
  <locale android:name="zh"/>
  <locale android:name="en"/>
</locale-config>
```

2.在清单中，添加一行指向这个新文件的代码：

```
<manifest
   ...
   <application
      ...
      android:localeConfig="@xml/locales_config">
   </application>
</manifest>
```

如何在设置中进行设置

用户可以通过新的系统设置为每个应用选择首选语言。他们可以通过以下两种方式访问这些设置：

通过系统设置访问

设置 > 系统 > 语言和输入法 > 应用语言 >（选择一款应用）

通过应用设置访问

设置 > 应用 >（选择一款应用）> 语言

处理应用内语言选择器

如需设置用户的首选语言，需要让用户在语言选择器中选择语言区域，然后在系统中设置该值：

```
val appLocale: LocaleListCompat = LocaleListCompat.forLanguageTags("xx-YY")
// 注意：需要在主线程上调用它，因为它可能需要Activity.restart()
AppCompatDelegate.setApplicationLocales(appLocale)
```

如需支持搭载 Android 12（S-32）及更低版本的设备，请在应用的 AppLocalesMetadataHolderService 服务的清单条目中将 autoStoreLocales 值设置为 true 并将 android:enabled 设置为 false，以指示 AndroidX 处理语言区域存储空间，如以下代码段所示：

```
<application
  ...
  <service
   android:name="androidx.appcompat.app.AppLocalesMetadataHolderService"
   android:enabled="false"
   android:exported="false">
   <meta-data
    android:name="autoStoreLocales"
    android:value="true" />
  </service>
  ...
</application>
```

请注意，将 autoStoreLocales 值设为 true 会导致主线程上出现阻塞读取，并可能会导致 StrictMode diskRead 和 diskWrite 违规行为。

/ 带主题的应用图标 /

这个功能其实官方已经宣传了挺久了，从 Android 13 起，用户可以选择启用带主题的应用图标。借助此功能，用户可以调节受支持的 Android 启动器中应用图标的色调，以继承所选壁纸和其他主题的配色。

如需支持此功能，必须提供自适应图标和单色应用图标两种，并通过 AndroidManifest.xml 中的 <adaptive-icon> 元素指向该单色应用图标。如果用户启用了带主题的应用图标，而启

动器支持此功能，则系统将使用用户选择的壁纸和主题来确定色调颜色，然后该颜色将应用于单色应用图标。

在以下任何情况下，主屏幕都不会显示带主题的应用图标，而是显示自适应或标准应用图标：

- 如果用户未启用带主题的应用图标
- 如果应用不提供单色应用图标
- 如果启动器不支持带主题的应用图标

单色应用图标

- 应是一个 VectorDrawable 也就是矢量图。
- 徽标适合 108 x 108 dp 容器中的 44 x 44 dp 的区域内。如果需要更大尺寸的徽标，最大可以为 72 x 72 dp。（这里所说的徽标就是图标中代表应用的样子，类似于支付宝的“支”字）。
- 官方建议使用平面徽标，但如果徽标是三维的，可以使用 Alpha 渐变来实现。

下面是官方单色应用图标的图片展示：

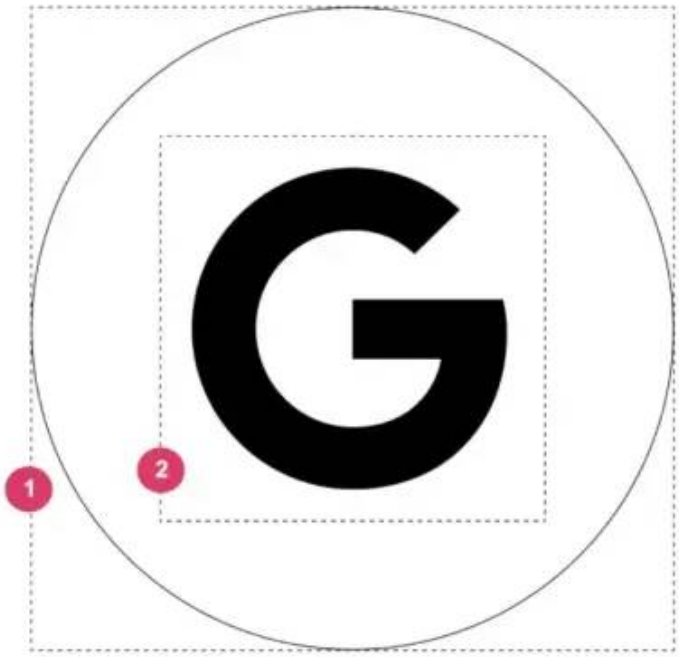


图 2：带主题的应用图标的测量值

1 容器区域 (108 x 108 dp)。

2 徽标区域（建议尺寸为 44 x 44 dp，最大尺寸为 72 x 72 dp）。

如何使用

将 monochrome android:drawable 属性添加到 <adaptive-icon> 元素中。例如，在 res/mipmap-anydpi-v26/ic_launcher.xml 中：

```
<adaptive-icon >
   <background android:drawable="..." />
   <foreground android:drawable="..." />
   <monochrome android:drawable="@drawable/myicon" />
</adaptive-icon>
```

然后在 AndroidManifest.xml 中，使用 android:icon 定义图标：

```
<application
   …
   android:icon="@mipmap/ic_launcher"
   …>
</application>
```

注意：如果清单中同时包含 android:roundIcon 和 android:icon，必须移除对 android:roundIcon 的引用，或者在由 android:roundIcon 属性定义的可绘制对象中提供单色图标。

/ 小结 /

其实 Android 13 中更新的内容远不止这些，比如在 Android 13 中优化了 TextView 的 hyphenation 、还增加了彩色的矢量字体、还增加了 Receiver 的安全性等等，我只是挑选了一些和普通开发者相关的更新来简单描述了下。

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

一个Android沉浸式状态栏上的黑科技

Android动画实战，腾讯课堂加载动画效果实现

欢迎关注我的公众号

学习技术或投稿

▼

长按上图，识别图中二维码即可关注

阅读原文

趁这个软件还没倒闭，我连夜用Python下载了所有壁纸...

Python学习交流

aardio + Python 可视化快速开发桌面程序，一键生成独立 EXE

aardio