Android 13运行时权限变更一览

原创 郭霖 郭霖 2022-08-16 08:00 发表于江苏

收录于合集
#android 8 #我的原创 88

▲

点击上方蓝字即可关注
关注后可查看所有经典文章

要不了多久，Android 13正式版就要发布了。

其实就在几个月前，我写了一篇关于Android 13首个开发者体验版的全面介绍，详情可以参考Android 13 Developer Preview一览 。

那么相比于首个开发者体验版，目前Android 13已经进入了平台稳定期阶段，也就是说API基本已经固定，不会再有什么大的修改了。

于是我又重新回顾了一遍Android 13的重要新特性和行为变更，发现有一处重大变化在首个开发者体验版中几乎没有提及，那就是Android 13的运行时权限变更。

因此，今天我就再写一篇Android 13的运行时权限变更一览，带你全面了解Android 13的所有运行时权限变更。

/ 细化的媒体权限 /

Google在Android 13上对本地数据访问权限做了更进一步的细化。

只能说Google为了保护用户隐私已经不遗余力了，而且今天的这步棋其实已经提前布局了很久了。

要知道，早在Android 10系统中，Google就禁用了本地文件通过绝对路径直接访问的形式，而是要通过MediaStore API来进行访问，我们称这个功能为Scoped Storage。

关于Scoped Storage，我在两年前就写过一篇文章进行介绍，详细请参考 Android 10适配要点，作用域存储 。

在这篇文章中，有这样的一处描述：

> Android 10系统针对文件类型进行了分类，图片、音频、视频这三类文件将可以通过MediaStore API来进行访问，而其他类型的文件则需要使用系统的文件选择器来进行访问。
>
> 另外，我们的应用程序向媒体库贡献的图片、音频或视频，将会自动拥有其读写权限，不需要额外申请READ_EXTERNAL_STORAGE和WRITE_EXTERNAL_STORAGE权限。而如果你要读取其他应用程序向媒体库贡献的图片、音频或视频，则必须要申请READ_EXTERNAL_STORAGE权限才行。WRITE_EXTERNAL_STORAGE权限将会在未来的Android版本中废弃。

这部分描述在Android 13之前看起来基本都是正确的。WRITE_EXTERNAL_STORAGE权限虽然还没有被废弃，但是我们无论在各种场景下几乎都已经不太可能再用到它了。

然而在Android 13当中，Google为了让用户能够更精细化地管理媒体权限，反而先对READ_EXTERNAL_STORAGE权限下手了。

从Android 13开始，如果你的应用targetSdk指定到了33或以上，那么READ_EXTERNAL_STORAGE权限就完全失去了作用，申请它将不会产生任何的效果。

与此相对应的，Google新增了READ_MEDIA_IMAGES、READ_MEDIA_VIDEO和READ_MEDIA_AUDIO这3个运行时权限，分别用于管理手机的照片、视频和音频文件。

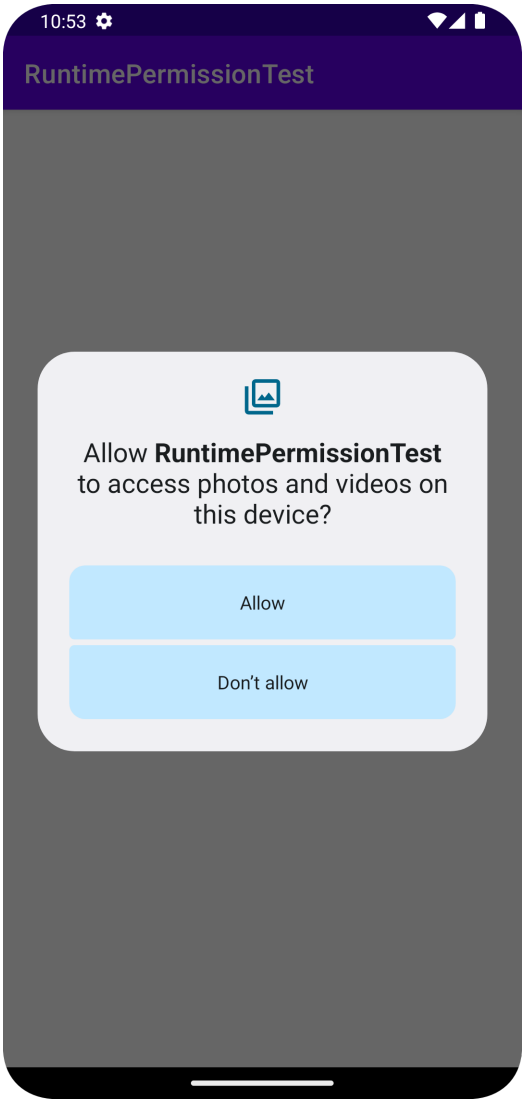
也就是说，以前只要申请一个READ_EXTERNAL_STORAGE权限就可以了。现在不行了，得按需申请，用户从而能够更加精细地了解你的应用到底申请了哪些媒体权限。

至于申请运行时权限的代码都是同样的模板，并没有什么特别的地方。这里给大家贴出一个用Activity Result API申请的版本，PermissionX由于我还没去适配Android 13，暂时还没法跟大家演示，等适配完成后我会再写一篇文章。

```
class MainActivity : AppCompatActivity() {

    private val requestPermissionLauncher = registerForActivityResult(
        ActivityResultContracts.RequestPermission()) { granted ->
        if (granted) {
            // User allow the permission.
        } else {
            // User deny the permission.
        }
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val requestBtn = findViewById<Button>(R.id.request_btn)
        requestBtn.setOnClickListener {
            if (Build.VERSION.SDK_INT >= 33) {
                requestPermissionLauncher.launch(Manifest.permission.READ_MEDIA_IMAGES)
            }
        }
    }
}
```

可以看到，这里使用ActivityResult API来申请了READ_MEDIA_IMAGES权限。如果你还没有了解过Activity Result API的朋友，可以参考这篇文章Activity Result API详解，是时候放弃startActivityForResult了 。

运行效果如下图所示：



比较奇怪的是，这里我在代码中只申请了读取照片的权限，但是截图上却显示我们正在申请读取照片和视频的权限。并且我在本地进行了验证，这两个权限确实是会一同授予的。而音频权限则不会和它们一同授予，还需要单独申请才行。

我的猜想是，这两个权限都属于同一个权限组，所以只要其中一个授予了，另外一个权限也就自动授予了。但是我在官方文档上没有找到对此的任何说明，所以在编写代码时请不要基于此行为去做任何的业务逻辑，因为权限组Google是随时都可能调整的，我们还是应该按照自己的业务需求，按需申请权限才对。

另外，为了考虑向下兼容性，我们在AndroidManifest.xml中声明权限时应该这样写：

```
<manifest ...>
    <!-- Required only if your app targets Android 13. -->
```

```
    <!-- Declare one or more the following permissions only if your app needs
    to access data that's protected by them. -->
    <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>
```

可以看到，API 32也就是Android 12及以下系统，我们仍然声明的是READ_EXTERNAL_STORAGE权限。从Android 13开始，我们就会使用READ_MEDIA_IMAGES、READ_MEDIA_VIDEO、READ_MEDIA_AUDIO来替代了。

在代码中申请权限时也应该做出同样的逻辑处理才行，这里就不再贴出了。

/ 通知运行时权限 /

通知运行时权限可以说是Android 13的重磅功能之一。这么多年过去了，Google终于将通知纳入了运行时权限管理。

其实我对通知是比较无感的，主要是因为Google太喜欢在通知上面做文章了。几乎每年的I/O大会，一定会有一个主题是专门讲通知新特性的，时间久了我对通知的变更已经基本免疫，实在学不动了。

但是，今年的变更却是不得不学，因为再不学的话，你的通知都要发不出去了。

通知栏真是一个让人又爱又恨的东西，这句话我相信不需要多做解释，用Android手机的人应该都懂。

在之前的Android系统中，任何一个应用想要发出通知的话都是不需要经过用户同意的，想发就能发。这就使得我们的手机通知栏经常被一些垃圾通知占领，真正重要的通知反而可能很难被找到。

为了解决这个问题，Google做出过很多改变与调整。比如说Android 8.0时加入的通知渠道，就是为了帮助用户更好地过滤有用通知和垃圾通知，具体可以参考这篇文章 一起来学习Android 8.0系统的通知栏适配吧 。

但通知渠道的加入，也只是让用户可以更加方便地筛选出那些不感兴趣的无用通知和垃圾通知，并予以屏蔽。本质上每个应用程序还是可以在完全不经用户同意的情况下随意发送通知。

而这次Android 13则把通知纳入了运行时权限管理，也就是说，以后想要发送通知，得要先经过用户同意授权才行了。

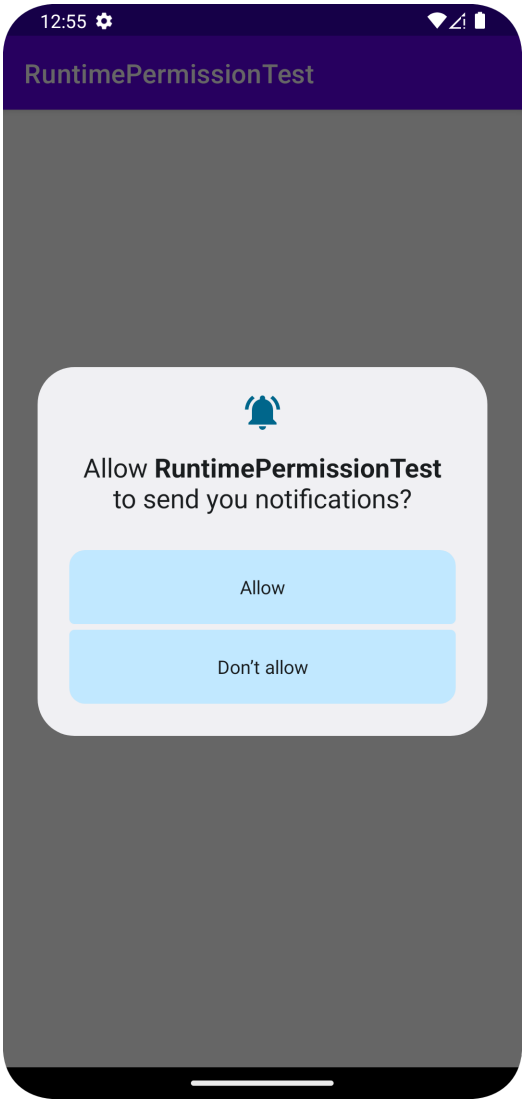
先说一下怎样在Android 13上申请发送通知权限吧，其实和一般的运行时权限并没有什么两样。首先在AndroidManifest.xml中对发送通知权限进行声明：

```
<manifest ...>
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
    <application ...>
        ...
    </application>
</manifest>
```

然后我们再次使用Activity Result API来请求权限即可：

```
class MainActivity : AppCompatActivity() {

    private val requestPermissionLauncher = registerForActivityResult(
        ActivityResultContracts.RequestPermission()) { granted ->
        if (granted) {
            // User allow the permission.
        } else {
            // User deny the permission.
        }
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val requestBtn = findViewById<Button>(R.id.request_btn)
        requestBtn.setOnClickListener {
            if (Build.VERSION.SDK_INT >= 33) {
                requestPermissionLauncher.launch(Manifest.permission.POST_NOTIFICATIONS)
            }
        }
    }
}
```

运行效果如下图所示：



从用法上讲，申请发送通知权限和其他运行时权限并无差别，但仍有不少细节是值得我们注意的。

其中一个必须要注意的点，POST_NOTIFICATIONS权限只有在应用程序的targetSdk指定成33或更高时才会有用。

当targetSdk为32及以下时，系统会认为你还没有为Android 13做好适配工作，此时申请POST_NOTIFICATIONS权限将不会产生任何效果。相对应地，它会在你首次创建通知渠道时弹出一个如上图所示的对话框。

而如果用户在此时选择了Don't allow，就将没有机会再次看到这个对话框了，也就是用户永久拒绝了我们发送通知的权限。直到以下两个情况发生：

- 用户卸载并重新安装了我们的应用。
- 我们将targetSdk升级到了33或更高。

另外，当用户的手机从Android 12升级到了Android 13，已安装应用的发送通知能力并不会发生变化。

也就是说，如果用户在Android 12上将我们应用的通知给屏蔽了，那么该设备升级到Android 13时，我们的应用也不会拥有发送通知权限。

但只要用户在Android 12上没有明确屏蔽我们应用的通知，那么该设备升级到Android 13后，我们的应用将会自动被授予发送通知权限。

最后，如果要判断一个运行时权限有没有被授权，通常情况下都可以这样写：

```
if (ContextCompat.checkSelfPermission(this, Manifest.permission.POST_NOTIFICATIONS)
    == PackageManager.PERMISSION_GRANTED) {
    // Permission granted
} else {
    // Permission not granted
}
```

但是这种写法在判断发送通知权限上面有一个明显的弊端，因为它只能在Android 13上使用，Android 13以下的系统是没有POST_NOTIFICATIONS权限的。

所以如果只是为了判断我们的应用现在有没有能力发出通知让用户看到，可以使用如下的写法，将保证在各个系统版本上都是能正常工作的：

```
val notificationManager = getSystemService(NOTIFICATION_SERVICE) as NotificationManager
if (notificationManager.areNotificationsEnabled()) {
    // Permission granted
} else {
    // Permission not granted
}
```

那么如果我们检测到发送通知没有被授权，同时用户还将这个权限永久拒绝了，该怎么办呢？

这个话题我准备留到PermissionX升级支持Android 13的时候，专门再写一篇文章进行介绍。

/ 其他新增权限 /

Android 13上最需要我们关注的新增权限就是以上这些，但它们并不是全部。

还有一些比较小众的新增权限可能大家用到的机会很少，这里就简单概括一下吧。

去年，Google在Android 12当中新增了几个蓝牙相关的运行时权限。原因是因为当开发者去访问一些蓝牙相关的接口时，却需要申请地理位置权限才行，这就让一些对隐私敏感的用户非常反感。

这是一个历史遗留问题，为了更好地保护用户隐私，Google在Android 12当中增加了BLUETOOTH_SCAN，BLUETOOTH_ADVERTISE，BLUETOOTH_CONNECT，这3个运行时权限。这样当开发者需要访问蓝牙相关的接口时，只需要请求这些蓝牙权限即可。

而在今年的Android 13当中，Google将保护用户隐私延伸到了WIFI领域。

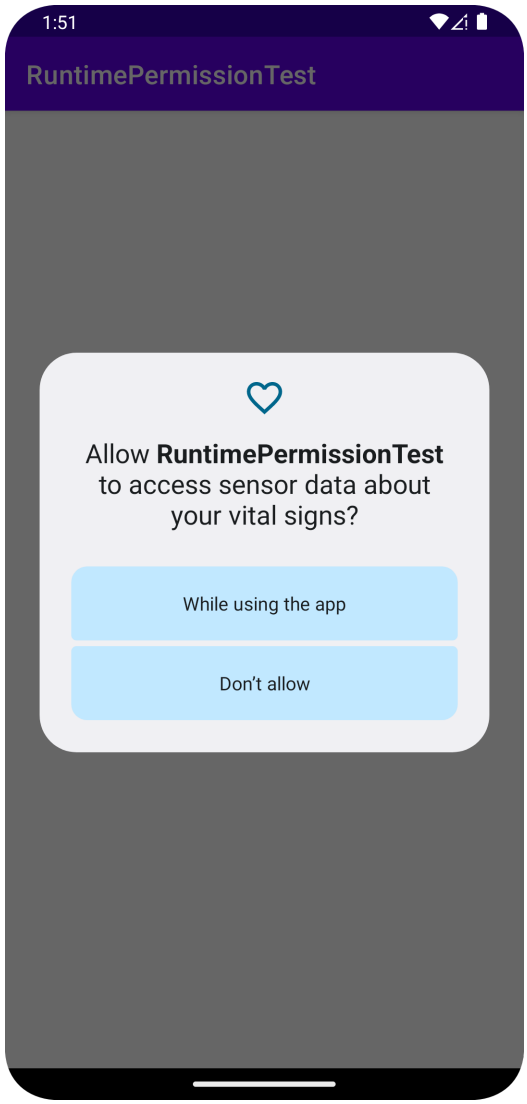
和蓝牙类似，当开发者去访问一些WIFI相关的接口时，如热点、WIFI直连、WIFI RTT等，也需要申请地理位置权限才行。

这其实也是一个历史遗留问题，用户肯定无法理解为什么使用一些WIFI功能时却需要授权地理位置权限。

为此，Android 13当中新增了一个NEARBY_WIFI_DEVICES权限，当再使用以上场景相关的WIFI API时，我们只需申请NEARBY_WIFI_DEVICES权限即可，从而更好地保护了用户的隐私。

另外还有一个变化是运动传感器权限。

之前我们如果想要读取手机运动传感器的数据，需要申请BODY_SENSORS权限。而在Android 13当中，Google给BODY_SENSORS权限又添加了一个只能在前台使用的限定。



可以看到，在Android 13上申请BODY_SENSORS权限时，用户只能授权在前台使用。

那么如果我们的应用程序就是要在后台获取运动传感器数据怎么办呢？别担心，Android 13又新增了一个BODY_SENSORS_BACKGROUND权限，申请这个权限即可。

需要注意的是，申请BODY_SENSORS_BACKGROUND权限之前必须得要先获得BODY_SENSORS授权才行，不然申请就是无效的。这个设定有点像当初Android 10增加后台获取地理位置权限的设定。

好了，以上就是Android 13运行时权限变更一览，希望对大家有所帮助。

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

Android 13 Developer Preview一览

模仿Android微信小程序，实现小程序独立任务视图的效果

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

收录于合集 #android 8

上一篇

PermissionX 1.7发布，全面支持Android 13运行时权限

下一篇

模仿Android微信小程序，实现小程序独立任务视图的效果

阅读原文

CSDN

MAUI安卓使用日历控件CalendarView

Xamarin Library