

码牛学院AndroidR训练营01-为什么需要适配AndroidR

背景: 码牛学院只为更好的你, Android11训练营预习资料, 让你彻底理解Android11新特性

问题一、Android 10的分区存储究竟影响了什么?

相信每一台Android手机的外部存储根目录都是乱得一塌糊涂, 这是因为在Android 10以前, 只要程序获得了`READ_EXTERNAL_STORAGE`权限, 就可以随意读取取外部存储的公有目录; 只要程序获得了`WRITE_EXTERNAL_STORAGE`权限, 就可以随意在写入 外部存储的公有目录上新建文件夹或文件。

于是Google终于开始动手了, 在Android 10中提出了**分区存储**, 意在限制程序对外部存储中公有目录的为所欲为。**分区存储对 内部存储私有目录 和 外部存储私有目录 都没有影响。**

简而言之, 在Android 10中, 对于私有目录的读写没有变化, 仍然可以使用File那一套, 且不需要任何权限。而对于公有目录的读写, 则必须使用 `MediaStore` 提供的API或是 `SAF` (存储访问框架), 意味着我们不能再使用File那一套来随意操作公有目录了。

在Android 11中, 没有了之前的兼容模式, 不能用File的那一套, 意味着以前访问 和读取App外置卡的目录失效了

使用分区存储的应用对自己创建的文件始终拥有读/写权限, 无论文件是否位于应用的私有目录内。因此, 如果您的应用仅保存和访问自己创建的文件, 则无需请求获得

`READ_EXTERNAL_STORAGE` 或 `WRITE_EXTERNAL_STORAGE` 权限。

若要访问其他应用创建的文件, 则需要 `READ_EXTERNAL_STORAGE` 权限。并且仍然只能使用 `MediaStore` 提供的API或是 `SAF` (存储访问框架) 访问。

需要注意的是, `MediaStore` 提供的API只能访问: 图片、视频、音频, 如果需要访问其它任意格式的文件, 需要使用 `SAF` (存储访问框架), 它会调用系统内置的文件浏览器供用户自主选择文件。

问题二、必须要适配吗?

本来从Android 10开始, Google就决定强制采用分区存储, 但从预览版的反馈来看, 很多应用都GG了, 因此Google决定给开发者一段过渡时间, 暂时不强制要求, 但早晚会强制要求的。

如果我们将 `targetSdkVersion` 设置为低于29的值, 那么即使不做任何关于Android 10的适配, 我们的项目也可以成功运行到Android 10手机上。

如果我们将 `targetSdkVersion` 设置为29了, 但就是不想适配分区存储, 可以在清单文件中做如下设置:

```
<manifest ... >
    <application android:requestLegacyExternalStorage="true" ... >
        ...
    </application>
</manifest>
```

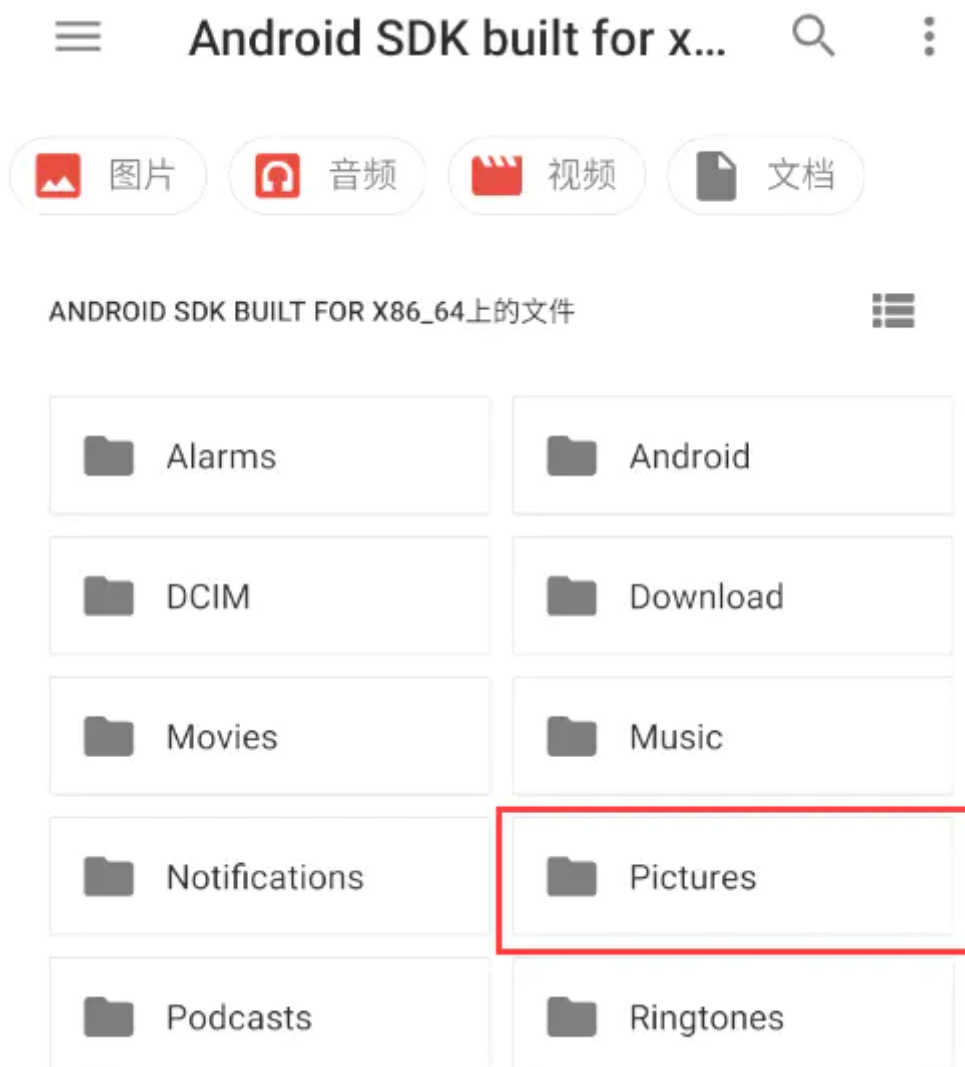
问题四、那我应该怎么操作外部存储?

现在我们已经知道, 在应用确认支持分区存储之后, 就不能再使用以前那一套来操作外部存储了。那现在应该怎么办呢?

1. Google官方推荐我们使用 `MediaStore` 提供的API访问图片、视频、音频资源,
2. 使用 `SAF` (存储访问框架) 访问其它任意类型的资源。

问题五, 如何保存目录在Pictures目录 使用MediaStore将图片保存到Pictures目录

在 `Environment` 中我们能找到很多公有目录文件夹的名字, 其中 `Pictures` 这个文件夹就适合用来保存图片数据:



在以前, 我们经常会根据目录或文件的绝对路径得到File对象, 再将File对象传给 `FileOutputStream` 得到输出流, 然后就可以愉快地写入数据了。Android 10以后, 我们要向这些公有目录写入数据, 必须要用 `MediaStore` 了。

下面, 我们通过代码学习如何将Bitmap保存到Pictures文件夹下:

```
const val APP_FOLDER_NAME = "ExternalScopeTestApp"

val bitmap = BitmapFactory.decodeResource(resources, R.drawable.die)
val displayName = "${System.currentTimeMillis()}.jpg"
val mimeType = "image/jpeg"
val compressFormat = Bitmap.CompressFormat.JPEG

private fun saveBitmapToPicturePublicFolder(
    bitmap: Bitmap,
    displayName: String,
    mimeType: String,
```

```

        compressFormat: Bitmap.CompressFormat
    ) {
        val contentValues = ContentValues()
        contentValues.put(MediaStore.MediaColumns.DISPLAY_NAME, displayName)
        contentValues.put(MediaStore.MediaColumns.MIME_TYPE, mimeType)
        val path = getAppPicturePath()
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
            contentValues.put(MediaStore.MediaColumns.RELATIVE_PATH, path)
        } else {
            val fileDir = File(path)
            if (!fileDir.exists()){
                fileDir.mkdir()
            }
            contentValues.put(MediaStore.MediaColumns.DATA, path + displayName)
        }
        val uri =
            contentResolver.insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
            contentValues)
        uri?.also {
            val outputStream = contentResolver.openOutputStream(it)
            outputStream?.also { os ->
                bitmap.compress(compressFormat, 100, os)
                os.close()
                Toast.makeText(this, "添加图片成功", Toast.LENGTH_SHORT).show()
            }
        }
    }
}

fun getAppPicturePath(): String {
    return if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
        // full path
        "${Environment.getExternalStorageDirectory().absolutePath}/" +
            "${Environment.DIRECTORY_PICTURES}/${APP_FOLDER_NAME}/"
    } else {
        // relative path
        "${Environment.DIRECTORY_PICTURES}/${APP_FOLDER_NAME}/"
    }
}
}

```

5.2、如何读取 在Pictures目录的图片 使用MediaStore获取媒体库中的图片

我们向Pictures中添加了图片文件，怎么才能获取到呢？也必须通过MediaStore。如果我们没有获得存储空间权限，那么我们只能通过MediaStore获取到自己应用创建的图片；如果我们获取了存储空间权限，那么我们就可以获取到其它应用创建的图片了。

我们通过代码来学习：

```

val cursor = contentResolver.query(
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
    null,
    null,
    null,
    "${MediaStore.MediaColumns.DATE_ADDED} desc"
)

cursor?.also {
    while (it.moveToNext()) {

```

```
        val id =
it.getLong(cursor.getColumnIndexOrThrow(MediaStore.MediaColumns._ID))
        val displayName =

it.getString(cursor.getColumnIndex(MediaStore.MediaColumns.DISPLAY_NAME))
        val uri =
ContentUris.withAppendedId(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
id)
    }
}
cursor?.close()
```

代码很简单，最终通过Uri与图片文件id的组合，得到了图片文件的Uri。得到了这个图片的Uri后，怎么显示出来呢？可以使用Glide，因为Glide原生支持Uri：

```
Glide.with(this).load(uri).into(ivPicture)
```

如果没有使用Glide呢？可以这样做，得到一个Bitmap：

```
val openFileDescriptor = contentResolver.openFileDescriptor(uri, "r")
openFileDescriptor?.apply {
    val bitmap = BitmapFactory.decodeFileDescriptor(fileDescriptor)
    ivPicture.setImageBitmap(bitmap)
}
openFileDescriptor?.close()
```

但是需要注意，如果图片分辨率高，bitmap会很占用内存，而实际要显示的区域可能比原图小得多，需要自己控制下bitmap的像素。

在没有获取存储空间权限对情况下，我们只能获取到应用自己创建的图片：

ExternalScopedTest

是否已经获取存储空间权限：否

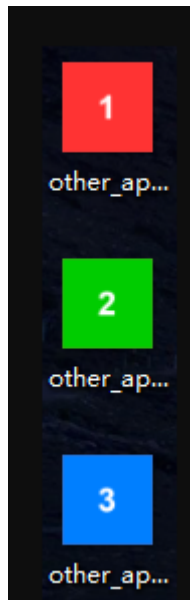
请求存储空间权限

使用MediaStore添加图片

使用MediaStore获取所有(权限)/本应用图片
删除非本应用图片(权限)

使用SAF选择PDF文件

下面我们预先在手机中放置几张图片，模拟它们是其它应用创建的，这几张图片如下：



它们有两张在DCIM/Camera，有一张在下载/OtherApp_01目录下。现在我们授予应用存储权限，看看是否能获取到这三张图片：

ExternalScopedTest

是否已经获取存储空间权限：否

请求存储空间权限

使用MediaStore添加图片

使用MediaStore获取所有(权限)/本应用图片
删除非本应用图片(权限)

使用SAF选择PDF文件

果然显示出来了。大家可能有一个疑问：应用是怎么在拥有权限的情况下只显示自己保存的图片的？不是一旦有权限后，就是查询的所有吗？答案就是增加WHERE语句，过滤DATA和RELATIVE_PATH

```
private fun queryImages(queryAll: Boolean = false): List<ImageBean> {
```

```

var pathKey = ""
var pathValue = ""
if (!queryAll) {
    pathKey = if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
        MediaStore.MediaColumns.DATA
    } else {
        MediaStore.MediaColumns.RELATIVE_PATH
    }
    // RELATIVE_PATH会在路径的最后自动添加/
    pathValue = getAppPicturePath()
}
val dataList = mutableListOf<ImageBean>()
val cursor = contentResolver.query(
    MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
    null,
    if (pathKey.isEmpty()) {
        null
    } else {
        "$pathKey LIKE ?"
    },
    if (pathValue.isEmpty()) {
        null
    } else {
        arrayOf("%$pathValue%")
    },
    "${MediaStore.MediaColumns.DATE_ADDED} desc"
)

cursor?.also {
    while (it.moveToNext()) {
        val id =
it.getLong(cursor.getColumnIndexOrThrow(MediaStore.MediaColumns._ID))
        val displayName =
it.getString(cursor.getColumnIndex(MediaStore.MediaColumns.DISPLAY_NAME))
        val uri =
ContentUris.withAppendedId(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, id)
        dataList.add(ImageBean(id, uri, displayName))
    }
}
cursor?.close()
return dataList
}

```

用LIKE的原因是DATA中存储的是图片的绝对路径，我们需要匹配应用自己图片路径下的所有图片。到此，获取媒体库中的图片就学习完毕。

5.3、如何删除图片 使用MediaStore删除媒体库中的图片

同样的，删除自己创建的图片不需要任何权限，但是删除或者修改其它应用创建的图片就需要权限了，而且即使我们拥有了存储权限，也不能修改或删除其它APP的资源，

需要由 `MediaProvider` 弹出弹框给用户选择是否允许APP修改或删除图片、视频、音频文件。用户操作的结果，将通过 `onActivityResult` 回调返回到APP。如果用户允许，APP将获得该uri的修改权限，直到设备下一次重启。我们先来学习删除自己应用的图片：


```
val row = contentResolver.delete(imageUri, null, null)
if (row > 0) {
    Toast.makeText(this, "删除成功", Toast.LENGTH_SHORT).show()
}
```

很简单，图片从 `ContentResolver` 中查询出来的时候，我们可以获取到id，图片的uri就是通过 `MediaStore.Images.Media.EXTERNAL_CONTENT_URI` 与图片的id组合而来。现在只需要通过uri进行删除即可。

我们来看下在没有存储权限时，删除应用创建图片的效果，当然如果有存储权限，也是一样的：

ExternalScopedTest

是否已经获取存储空间权限：否

请求存储空间权限

使用MediaStore添加图片

使用MediaStore获取所有(权限)/本应用图片
删除非本应用图片(权限)

使用SAF选择PDF文件

但是这段代码是不够严谨的，因为当我们删除的是其它应用的资源，程序会闪退，并抛出：`RecoverableSecurityException` 异常。因此我们需要捕获这个异常，提示用户给予此uri修改或删除的权限：

```
companion object {
```

```

        const val REQUEST_DELETE_PERMISSION = 1
    }

    private var pendingDeleteImageUri: Uri? = null
    private var pendingDeletePosition: Int = -1

    private fun deleteImage(imageUri: Uri, adapterPosition: Int) {
        var row = 0
        try {
            // Android 10+中,如果删除的是其它应用的Uri,则需要用户授权
            // 会抛出RecoverableSecurityException异常
            row = contentResolver.delete(imageUri, null, null)
        } catch (securityException: SecurityException) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
                val recoverableSecurityException =
                    securityException as? RecoverableSecurityException
                ?: throw securityException
                pendingDeleteImageUri = imageUri
                pendingDeletePosition = adapterPosition
                // 我们可以使用IntentSender向用户发起授权

                requestRemovePermission(recoverableSecurityException.userAction.actionIntent.intentSender)
            } else {
                throw securityException
            }
        }

        if (row > 0) {
            Toast.makeText(this, "删除成功", Toast.LENGTH_SHORT).show()
            pictureAdapter.deletePosition(adapterPosition)
        }
    }

    private fun requestRemovePermission(intentSender: IntentSender) {
        startIntentSenderForResult(intentSender, REQUEST_DELETE_PERMISSION,
            null, 0, 0, 0, null)
    }

    private fun deletePendingImageUri(){
        pendingDeleteImageUri?.let {
            pendingDeleteImageUri = null
            deleteImage(it,pendingDeletePosition)
            pendingDeletePosition = -1
        }
    }

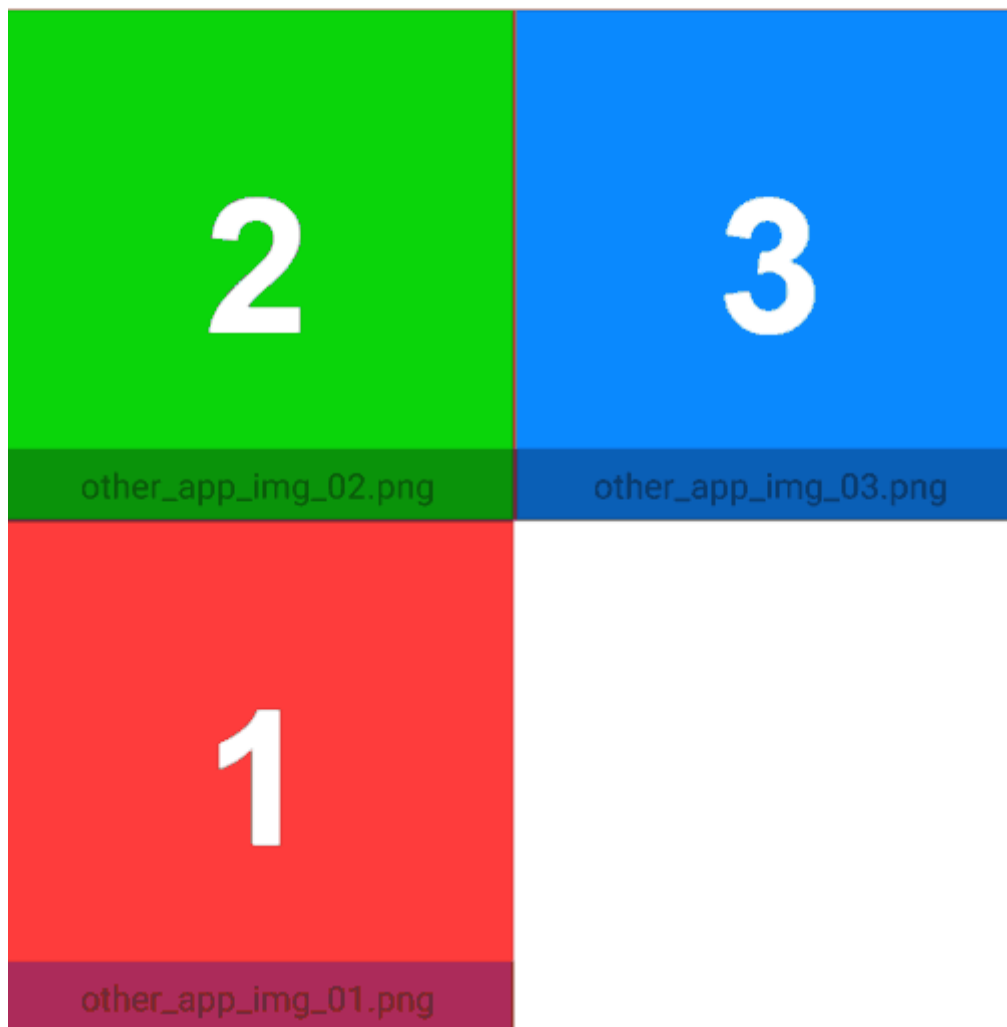
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?)
    {
        super.onActivityResult(requestCode, resultCode, data)
        if (resultCode == Activity.RESULT_OK &&
            requestCode == REQUEST_DELETE_PERMISSION
        ) {
            // 执行之前的删除逻辑
            deletePendingImageUri()
        }
    }
}

```

简单解释下，当修改或删除非本应用创建的文件uri时，在Android 10+的系统中，会抛出 `RecoverableSecurityException`，我们捕获到这个异常后，从异常中获得了 `IntentSender`，并使用它来向用户索取该uri的修改、删除权限。代码都很简单，不再赘述，效果如下：

ExternalScopedTest

☑ 显示手机媒体库中所有图片(需要权限)



至此，删除媒体库中的图片就学习完成了。

问题六、我想读取Download文件夹下的某个非媒体文件怎么办？

拿PDF举例，显然，PDF不属于音频、视频、图片，因此我们不能使用 `MediaStore` 来获取。对于这种其它类型的文件，我们一般使用 `SAF`（存储访问框架）让用户选择：

```
private fun selectPdfusesSAF() {
    val intent = Intent(Intent.ACTION_OPEN_DOCUMENT).apply {
        type = "application/pdf"
        // 我们需要使用ContentResolver.openFileDescriptor读取数据
        addCategory(Intent.CATEGORY_OPENABLE)
    }
    startActivityForResult(intent, REQUEST_OPEN_PDF)
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?)
{
    super.onActivityResult(requestCode, resultCode, data)
    when (requestCode) {
        REQUEST_OPEN_PDF -> {
            if (resultCode == Activity.RESULT_OK) {
                data?.data?.also { documentUri ->
                    val fileDescriptor =
                        contentResolver.openFileDescriptor(documentUri, "r") ?:
                }
                return
            }
            // 现在,我们可以使用PdfRenderer等类通过fileDescriptor读取pdf内容
            Toast.makeText(this, "pdf读取成功",
                Toast.LENGTH_SHORT).show()
        }
    }
}
```

注意，`ACTION_OPEN_DOCUMENT` 用于打开文件。

问题七、我想创建任意类型文件怎么办？

也是使用 `SAF`（存储访问框架）让用户去创建。其Intent Action为：

`ACTION_CREATE_DOCUMENT`，我们可以使用Intent的`putExtra()`来指定文件的名字：

`intent.putExtra(Intent.EXTRA_TITLE, "Android.pdf")`，有点类似于“另存为”功能。其它的使用方法差不多，就不多说了。

问题八、我想将文件下载到Download目录怎么办？

拿下载app为例，在Android 10以前，只要获取到了File对象，就能得到输入流，而由于我们适配了Android 10中的分区存储，因此不能这样做了。`MediaStore` 中提供了一种Downloads集合，专门用于执行文件下载操作。它的使用和添加图片是几乎一样的

```
private fun downloadApkAndInstall(fileurl: String, apkName: String) {
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.Q) {
        Toast.makeText(this, "请使用原始方式", Toast.LENGTH_SHORT).show()
        return
    }
    thread {
        try {
            val url = URL(fileurl)
            val connection = url.openConnection() as HttpURLConnection
            connection.requestMethod = "GET"
        }
    }
}
```

```

        val inputStream = connection.inputStream
        val bis = BufferedInputStream(inputStream)
        val values = ContentValues()
        values.put(MediaStore.MediaColumns.DISPLAY_NAME, apkName)
        values.put(MediaStore.MediaColumns.RELATIVE_PATH,
getAppDownloadPath())
        val uri =
contentResolver.insert(MediaStore.Downloads.EXTERNAL_CONTENT_URI, values)
        uri?.also {
            val outputStream = contentResolver.openOutputStream(uri) ?:
return@thread
            val bos = BufferedOutputStream(outputStream)
            val buffer = ByteArray(1024)
            var bytes = bis.read(buffer)
            while (bytes >= 0) {
                bos.write(buffer, 0, bytes)
                bos.flush()
                bytes = bis.read(buffer)
            }
            bos.close()
            runOnUiThread {
                installAPK(uri)
            }
        }
        bis.close()

    } catch (e: Exception) {
        e.printStackTrace()
    }
}

private fun installAPK(uri: Uri) {
    val intent = Intent(Intent.ACTION_VIEW)
    intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
    intent.setDataAndType(uri, "application/vnd.android.package-archive")
    startActivity(intent)
}

```

由于我们获取到的这个uri本来就是 `content://` 开头，所以不需要使用 `FileProvider`。对于应用私有目录的文件，我们可以使用 `FileProvider` 进行分享。