**FileProvider 在 Android N 上的应用**

**一、背景**

Android 从 N 开始不允许以 file:// 的方式通过 Intent 在两个 App 之间分享文件，取而代之的是通过 FileProvider 生成 content://Uri 。如果在 Android N 以上的版本继续使用 file:// 的方式分享文件，则系统会直接抛出异常，导致 App 出现 Crash ，同时会报以下错误日志：

FATAL EXCEPTION: main

Process: com.inthecheesefactory.lab.intent\_fileprovider, PID: 28905

android.os.FileUriExposedException: file:///storage/emulated/0/.../xxx/xxx.jpg exposed beyond app through ClipData.Item.getUri**()**

at android.os.StrictMode.onFileUriExposed**(**StrictMode.java:1799**)**

at android.net.Uri.checkFileUriExposed**(**Uri.java:2346**)**

at android.content.ClipData.prepareToLeaveProcess**(**ClipData.java:832**)**

当然如果工程的 targetSDK 小于24，暂时还不会遇到这个问题，一旦升级到24及以上，则会立即出现上述问题，所以提早做好预防很有必要，否则等到线上曝出大量的 bug 就很被动了。

**二、关于 FileProvider**

官方对于 FileProvider 的解释为：FileProvider 是一个特殊的 ContentProvider 子类，通过 content://Uri 代替 file://Uri 实现不同 App 间的文件安全共享。

当通过包含 Content URI 的 Intent 共享文件时，需要申请临时的读写权限，可以通过 Intent.setFlags() 方法实现。

而 file://Uri 方式需要申请长期有效的文件读写权限，直到这个权限被手动改变为止，这是极其不安全的做法。因此 Android 从 N 版本开始禁止通过 file://Uri 在不同 App 之间共享文件。

**三、FileProvider 的使用流程**

完成整个文件共享的流程，需要配置以下5点：

1. 定义一个 FileProvider
2. 指定有效的文件
3. 为文件生成有效的 Content URI
4. 申请临时的读写权限
5. 发送 Content URI 至其他的 App

**1. 定义 FileProvider**

FileProvider 已经把文件生成 Content URI 的工作帮我们做掉了，因此我们只需要在 AndroidManifest.xml 文件中配置 <provider> 元素并提供相应的属性。

重要的属性包括以下四个：

* 设置 android:name 为android.support.v4.content.FileProvider，这是固定的，不需要手动更改；
* 设置 android:authorities 为 application id + .provider ；
* 设置 android:exported 为 false ，表示 FileProvider 不是公开的；
* 设置 android:grantUriPermissions 为 true 表示允许临时读写文件。

**此处需要特别说明的是**

1. android:authorities 最好是 application id 而不能直接用包名硬编码，因为 Android 系统要求 android:authorities 对于每个 App 而言必须是唯一的。
2. 假如 FileProvider 用在 SDK 中，多个 App 都在调用同一个 SDK，而 SDK 中的 android:authorities 为硬编码，那么 App 之间的 authorities 就会出现冲突，会报 Install shows error in console: INSTALL FAILED CONFLICTING PROVIDER 的错误。
3. 如果 SDK 的 android:authorities 是 application id，那么 authorities 会和宿主 App 的 application id 保持一致，就不会出现 authorities 冲突的问题。
4. 在 Java 代码中调用 getPackageName() 返回的是 application id ，而非 package name ，要验证这一点也很容易，在 build.gradle 文件中定义和包名不同的 application id ，打印代码中 getPackageName() 的返回值，就会发现返回值是 build.gradle 中自定义的 application id ，而非 package name
5. 关于 package name 和 application id 的区别可以参考 [ApplicationId 与 PackageName 的区别](http://link.zhihu.com/?target=http%3A//blog.csdn.net/feelang/article/details/51493501)

以下是一个简单的示例：

<manifest>

...

<application>

...

<provider

android:name="android.support.v4.content.FileProvider"

android:authorities="${applicationId}.provider"

android:exported="false"

android:grantUriPermissions="true">

...

</provider>

...

</application>

</manifest>

需要说明的是 ${applicationId} 是占位符，Gradle 会替换成我们在 build.gralde 中定义的 applicationId "com.domain.example"，如果 build.gradle 文件中没有定义,那么 application id的默认值是 App 的 package name。

**2. 指定有效的文件**

在生成 Content URI 之前你还需要提前指定文件目录，通常的做法是在 res 目录下新建一个 xml 文件夹，然后创建一个 xml 文件，在此文件中指定共享文件的路径和名字，示例如下：

<paths xmlns:android="http://schemas.android.com/apk/res/android">

<external-path name="my\_images" path="images/"/>

...

</paths>

其中 name 属性和 path 属性必填， name 表示共享文件的名字， path 代表文件路径。

* external-path 代表文件位于手机外部存储空间，访问效果如同 Environment.getExternalStorageDirectory()；
* files-path 代表文件位于手机内部存储空间，访问效果如同 getFilesDir()；
* cache-path 代表文件位于手机内部缓存空间，访问效果如同 getCacheDir()。

xml 文件创建完成后，还需要在 manifest 文件的 <provider> 元素下完成相应的配置，假定 xml 文件命名为 file\_paths.xml ，示例如下：

<provider

android:name="android.support.v4.content.FileProvider"

android:authorities="${applicationId}.provider"

android:exported="false"

android:grantUriPermissions="true">

<meta-data

android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/file\_paths" />

</provider>

**3. 为共享文件生成 Content URI**

文件配置完成后还需要生成可以被其他 App 访问的 Content URI，可以直接调用 FileProvider 提供的 getUriForFile(File file) 方法，顾名思义，传入文件名称就可以得到相应的 Content URI 。需要访问该文件的 App 可以通过 ContentResolver.openFileDescriptor 得到一个 ParcelFileDescriptor 对象。

假定你想要共享一个图片文件，文件存放的位置为手机内部存储空间下的 images 文件夹，图片文件名字为 default\_name.jpg ，那么生成 Content URI 方式如下：

File imagePath **=** **new** File**(**getContext**().**getFilesDir**(),** "images"**);**

File newFile **=** **new** File**(**imagePath**,** "default\_image.jpg"**);**

Uri contentUri **=** getUriForFile**(**getContext**(),** "com.mydomain.provider"**,** newFile**);**

最后生成的 Content URI 为

content://com.domain.example.provider/images/default\_image.jpg.

**4. 申请临时读写文件权限**

上文已经提到 FileProvider 可以申请临时读写文件权限，以增强安全性，所以 Content URI 生成完成后，还需要申请临时访问权限。

通常直接通过 intent.setFlags 即可完成，具体的权限名称为：Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION 和 Intent.FLAG\_GRANT\_WRITE\_URI\_PERMISSION。

**5. 发送 Content URI 至其他的 App**

万事已备，只需要发送出去即可，通常都会使用 startActivityForResult 方法发送，可以在 onActivityResult 中获取其他 App 的处理结果，完成整个操作闭环。

**三、实用场景——手机照相**

在 Android N 之前的版本调用相机获取图片可以用如下代码实现：

*// 设置照片需要存储的位置*

photoPath **=** FileUtil**.**getImageFile**().**getPath**()**

Intent intent **=** **new** Intent**();**

*// 指定开启系统相机的Action*

intent**.**setAction**(**MediaStore**.**ACTION\_IMAGE\_CAPTURE**);**

intent**.**addCategory**(**Intent**.**CATEGORY\_DEFAULT**);**

*// 把文件地址转换成Uri格式*

Uri uri **=** Uri**.**parse**(**"file://" **+** photoPath**);**

intent**.**putExtra**(**MediaStore**.**EXTRA\_OUTPUT**,** uri**);**

activity**.**startActivityForResult**(**intent**,** requestCode**);**

如果要想在 Android N 及以上版本上不会出错，则必须将 file:// 形式替换成 content:// ，具体的代码如下：

Intent intent **=** **new** Intent**();**

intent**.**setAction**(**MediaStore**.**ACTION\_IMAGE\_CAPTURE**);**

*// 系统版本大于N的统一用FileProvider处理*

**if** **(**Build**.**VERSION**.**SDK\_INT **>=** Build**.**VERSION\_CODES**.**N**)** **{**

*// 将文件转换成content://Uri的形式*

Uri photoURI **=** FileProvider**.**getUriForFile**(**activity**,**

activity**.**getPackageName**()+** ".provider"**,**

**new** File**(**photoPath**));**

*// 申请临时访问权限*

intent**.**setFlags**(**Intent**.**FLAG\_ACTIVITY\_NEW\_TASK **|** Intent**.**FLAG\_GRANT\_READ\_URI\_PERMISSION

**|** Intent**.**FLAG\_GRANT\_WRITE\_URI\_PERMISSION**);**

intent**.**putExtra**(**MediaStore**.**EXTRA\_OUTPUT**,** photoURI**);**

**}** **else** **{**

intent**.**addCategory**(**Intent**.**CATEGORY\_DEFAULT**);**

Uri uri **=** Uri**.**parse**(**"file://" **+** photoPath**);**

intent**.**putExtra**(**MediaStore**.**EXTRA\_OUTPUT**,** uri**);**

**}**

activity**.**startActivityForResult**(**intent**,** requestCode**);**

需要注意的是 getPackageName() 返回值是 application id，关于 application id 上文已经解释过，此处不再重复。

**实用场景——微信朋友圈多图分享**

微信官方不支持朋友圈直接多图分享，Android 之前的版本由于没有强制限制 file:// 的使用，所以可以通过访问微信包名的方式实现朋友圈多图分享，但是Android N 之后这种“曲线救国”的方式就不行了。

先来看一下之前如何通过访问包名实现朋友圈多图分享，代码如下：

Intent intent **=** **new** Intent**();**

intent**.**setComponent**(new** ComponentName**(**"com.tencent.mm"**,** "com.tencent.mm.ui.tools.ShareToTimeLineUI"**));**

intent**.**setAction**(**"android.intent.action.SEND\_MULTIPLE"**);**

*// List存储多张图片地址*

ArrayList**<**Uri**>** localArrayList **=** **new** ArrayList**<>();**

**for** **(int** i **=** 0**,** size **=** localPicsList**.**size**();** i **<** size**;** i**++)** **{**

localArrayList**.**add**(**Uri**.**parse**(**"file:///" **+** localPicsList**.**get**(**i**)));**

**}**

intent**.**putParcelableArrayListExtra**(**"android.intent.extra.STREAM"**,** localArrayList**);**

intent**.**setType**(**"image/\*"**);**

intent**.**putExtra**(**"Kdescription"**,** desc**);**

context**.**startActivity**(**intent**);**

这种方式可以直接绕过微信官方 SDK 实现多图分享，无需手动选择图片，唯一的问题就是没有分享结果的回调，也就是说无法判断是否分享成功，这在大部分情况下依然是一种可以接受的方案。

但是如果 targetSDK 大于等于24，那么这项功能就无效了，原因就是 Android N 不允许 file://Uri 的方式在不同的 App 间共享文件，但是如果换成 FileProvider 的方式，经试验发现依然是无效的，所以在 Android N 上无法实现朋友圈直接多图分享。

s