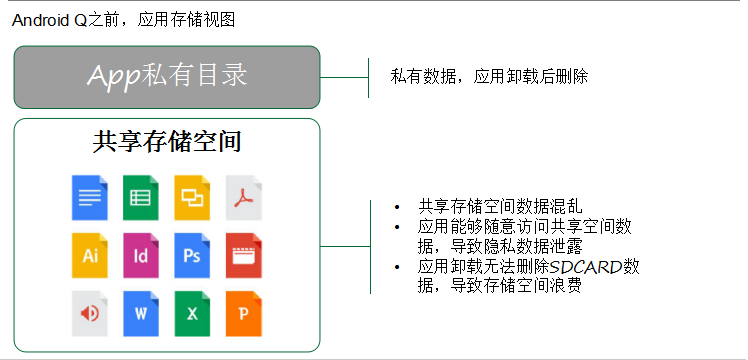
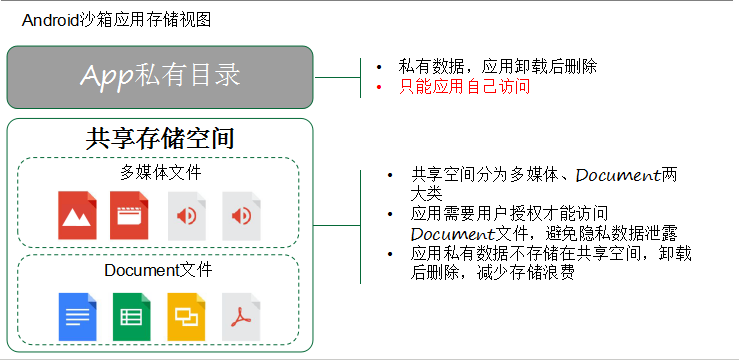
Android 沙箱模式介绍

# Scoped Storage

## 背景

为了让用户更好地控制自己的文件，并限制文件混乱情况，Android Q 更改了App访问设备存储空间的方式。





Android Q规定了App有两种存储空间模式视图：Legacy View、Filtered View。

* Legacy View（兼容模式）

兼容模式跟Android Q以前，App访问Sdcard一样，拥有完整的访问权限。

* Filtered View（沙箱模式）

App只能直接访问App-specific目录文件，没有权限访问App-specific外的文件。访问其他目录，只能通过MediaStore、SAF、或者其他App提供ContentProvider访问。

Scoped Storage将存储空间分为两部分：

* 公共目录：Downloads、Documents、Pictures 、DCIM、Movies、Music、Ringtones
  + 公共目录的文件在App卸载后，不会删除
  + 可以通过SAF、MediaStore接口访问
* App-specific目录
  + 对于Filtered View App，App-specific目录只能自己直接访问
  + App卸载，数据会清除。

## 兼容影响

Scoped Storage对于App访问存储方式、App数据存放以及App间数据共享，都产生很大影响。

## 适配

具体适配参考：

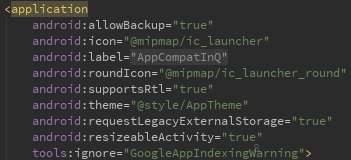
<https://developer.android.google.cn/preview/privacy/scoped-storage>

### 运行视图

#### App运行视图

系统通过下列确定App运行模式：

* App TargetSDK >= Q，默认Filtered View
* App TargetSDK < Q，声明了READ\_EXTERNAL\_STORAGE或者WRITE\_EXTERNAL\_STORAGE权限，默认Legacy View
* 应用可以通过AndroidManifest.xml，设置requestLegacyExternalStorage，选择对应的方式：

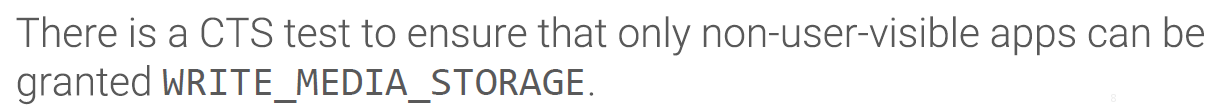


声明了READ\_EXTERNAL\_STORAGE、WRITE\_EXTERNAL\_STORAGE权限（没有声明则忽略）：

* + true表示Legacy View
  + false表示Filtered View
* 系统应用可以申请android.permission.WRITE\_MEDIA\_STORAGE系统权限，同样拥有完整存储空间权限，可以访问所有文件

但是这个在CTS测试中，会进行测试，只有没有用户交互、可见的App，才能申请。

具体参考《Android Bootcamp 2019 - Privacy Overview.pdf》。



* App在下列条件都成立时
  + 声明INSTALL\_PACKAGES、或者动态申请INSTALL\_PACKAGES权限
  + 拥有WRITE\_EXTERNAL\_STORAGE权限

App拥有外置存储空间Read、Write权限。但是通过Environment.isExternalStorageLegacy接口判断，返回不一定是Legacy View。

#### 判断当前App运行模式

判断当前App运行什么模式，可以通过这个API判断：

Environment.isExternalStorageLegacy();

### 读写公共目录

App启动Filtered View后，只能直接访问自身App-specific目录，所以Android Q，提供了两种访问公共目录的方法：

#### 通过MediaStore定义的Uri

MediaStore提供了下列几种类型的访问Uri，通过查找对应Uri数据，达到访问的目的。

下列每种类型又分为三种Uri，Internal、External、可移动存储:

* Audio
  + Internal: MediaStore.Audio.Media.INTERNAL\_CONTENT\_URI

content://media/internal/audio/media。

* + External: MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI

content://media/external/audio/media。

* + 可移动存储: MediaStore.Audio.Media.getContentUri

content://media/<volumeName>/audio/media。

* Video
  + Internal: MediaStore.Video.Media.INTERNAL\_CONTENT\_URI

content://media/internal/video/media。

* + External: MediaStore.Video.Media.EXTERNAL\_CONTENT\_URI

content://media/external/video/media。

* + 可移动存储: MediaStore.Video.Media.getContentUri

content://media/<volumeName>/video/media。

* Image
  + Internal: MediaStore.Images.Media.INTERNAL\_CONTENT\_URI

content://media/internal/images/media。

* + External: MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI

content://media/external/images/media。

* + 可移动存储: MediaStore.Images.Media.getContentUri

content://media/<volumeName>/images/media。

* File
  + MediaStore. Files.Media.getContentUri

content://media/<volumeName>/file。

* Downloads
  + Internal: MediaStore.Downloads.INTERNAL\_CONTENT\_URI

content://media/internal/downloads。

* + External: MediaStore.Downloads.EXTERNAL\_CONTENT\_URI

content://media/external/downloads。

* + 可移动存储: MediaStore.Downloads.getContentUri

content://media/<volumeName>/downloads。

##### 获取所有的Volume

对于前面描述的Uri中，getContentUri如何获取所有<volumeName>，可以通过下述方式：



##### Uri跟公共目录关系

MediaProvider对于App存放到公共目录文件，通过ContentResolver insert方法中Uri来确定，其中下表中<Uri路径>为相对路径，完整为：

content://media/<volumeName>/<Uri路径>。

|  |  |  |
| --- | --- | --- |
| Mine Type | Uri路径 | 一级目录 |
| audio/\* | images/media  images/media/# | Environment.DIRECTORY\_ALARMS  Environment.DIRECTORY\_MUSIC  Environment.DIRECTORY\_NOTIFICATIONS  Environment.DIRECTORY\_PODCASTS  Environment.DIRECTORY\_RINGTONES |
| audio/\* | audio/playlists  audio/playlists/# | Environment.DIRECTORY\_MUSIC |
| video/\* | video/media  video/media/# | Environment.DIRECTORY\_DCIM  Environment.DIRECTORY\_MOVIES |
| image/\* | images/media  images/media/# | Environment.DIRECTORY\_DCIM  Environment.DIRECTORY\_PICTURES |
| image/\* | video/thumbnails  video/thumbnails/# | Environment.DIRECTORY\_MOVIES |
| image/\* | images/thumbnails  images/thumbnails/# | Environment.DIRECTORY\_PICTURES |
| \*/\* | downloads  downloads/# | Environment.DIRECTORY\_DOWNLOADS |
| \*/\* | file  file/# | Environment.DIRECTORY\_DOWNLOADS  Environment.DIRECTORY\_DOCUMENTS |

##### 权限

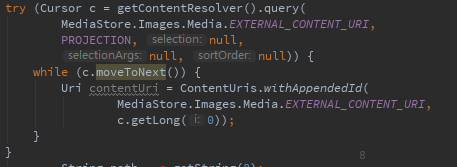
MediaStore通过不同Uri，为用户提供了增、删(如果通过File Uri无法删除文件，需要通过SAF接口)、改。

App对应的权限如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Audio | Image | Video | File | Downloads |
| WRITE\_EXTERNAL\_STORAGE | 能修改所有App新建的文件,前提是要要授权 | | | | |
| READ\_EXTERNAL\_STORAGE | 能读取所有App新建的文件，不能修改其他App新建文件 | | | | |
| 无 | 只能读取、修改自己新建的文件 | | | | |

##### 查询文件

通过ContentResolver，根据不同的Uri查询不同的内容：



*PS: MediaStore.Files 进行Query时候，只会显示图片、视频跟音频文件。*

##### 读取文件

通过ContentResolver query接口，查找出来文件后如何读取，可以通过下面的方式：

* 通过ContentResolver openFileDescriptor接口，选择对应的打开方式

例如”r”表示读，”w”表示写，返回ParcelFileDescriptor类型FD。

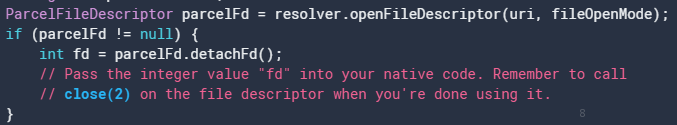
* 访问Thumbnail，通过ContentResolver loadThumbnail接口

通过传递大小，MediaProvider返回指定大小的Thumbnail。

* Native代码访问文件

如果Native代码需要访问文件，可以参考下面方式：

* + 通过openFileDescriptor返回ParcelFileDescriptor
  + 通过ParcelFileDescriptor.detachFd()读取FD
  + 将FD传递给Native层代码
  + App需要负责通过close接口关闭FD



##### 新建文件

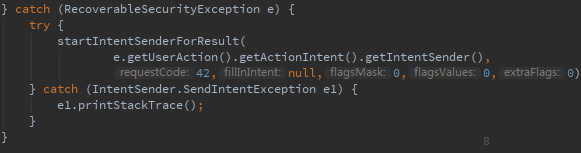
如果需要新建文件存放到公共目录，需要通过ContentResolver insert接口，使用不同的Uri，选择存储到不同的目录。



##### 修改文件

如果需要修改多媒体文件，需要通过ContentResolver query接口查找出来对应文件的Uri。

如果不是自己新建的文件，需要注意[2.3.2.1.3. 权限](#_权限)中描述，需要申请WRITE\_EXTERNAL\_STORAGE权限或者catch RecoverableSecurityException，弹框给用户选择。





通过下列接口，获取需要修改文件的FD或者OutputStream：

* getContentResolver().openOutputStream(contentUri)

获取对应文件的OutputStream。

* getContentResolver().openFile或者getContentResolver().openFileDescriptor

通过openFile或者openFileDescriptor打开文件，需要选择Mode为”w”，表示写权限。这些接口返回一个ParcelFileDescriptor。

getContentResolver().openFileDescriptor(contentUri,"w");

getContentResolver().openFile(contentUri,"w",null);

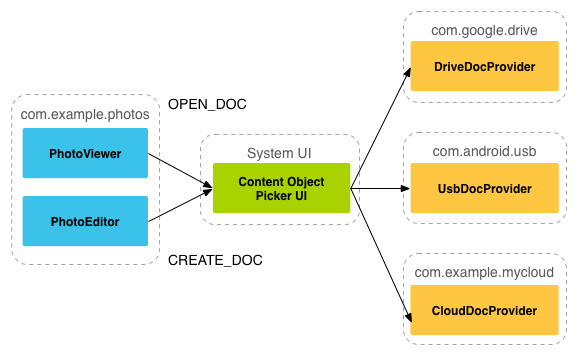
##### 删除文件

通过ContentResolver接口删除文件，Uri为query出来的Uri：

getContentResolver().delete(contentUri,null,null);

#### 通过SAF接口

SAF，即Storage Access Framework，通过选择不同的DocumentsProvider，提供给用户打开、浏览文件。

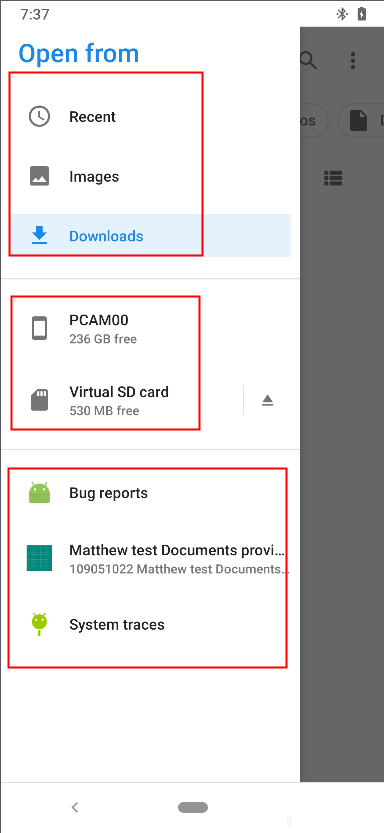


Android默认提供了下列DocumentsProvider：

MediaDocumentsProvider、ExternalStorageProvider、 DownloadStorageProvider。

他们之间差异是：

|  |  |  |  |
| --- | --- | --- | --- |
|  | MediaDocumentsProvider | ExternalStorageProvider | DownloadStorageProvider |
| 读 | 只能读取视频、音频、图片 | 全部内置、外置存储 | 读取Download目录 |
| 删除 | 可以删除 | | |
| 修改 | 无法修改 | 可以修改 | |



这个图片上，有三个区域，分别是：

* MediaDocumentsProvide
* DownloadStorageProvider
* ExternalStorageProvider
* 第三方DocumentsProvider

如何使用，具体参考：

<https://developer.android.com/guide/topics/providers/document-provider>

大致方法如下：

* 选择单个文件

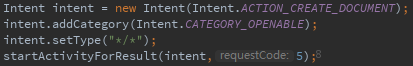


* 选择目录



文件管理程序，清理程序，可以通过这个方法获取对应目录以及子目录的全部管理权限。

* 新建文件



* 删除

DocumentsContract.deleteDocument(getContentResolver(),uri);

* 修改
  + 获取OutputStream

getContentResolver().openOutputStream(uri);

* + 获取可写ParcelFileDescriptor

getContentResolver().openFileDescriptor(contentUri,"w");

getContentResolver().openFile (contentUri,"w",null);

具体Demo参考：<https://github.com/android/storage>

### 访问App-specific目录

访问App-specific分为两种情况，第一是访问App自身App-specific目录，第二是访问其他App目录文件。

#### App自身App-specific目录

Android Q，App如果启动了Filtered View，那么只能直接访问自己目录的文件：

* Environment.getExternalStorageDirectory、getExternalStoragePublicDirectory

这些接口在Android Q上废弃，App是Filtered View，无法直接访问这个目录。

* 通过File(“/sdcard/”)访问

App是Filtered View，无法直接访问这个目录。

* 获取App-specific目录
  + 获取Media接口：getExternalMediaDirs
  + 获取Cache接口：getExternalCacheDirs
  + 获取Obb接口：getObbDirs
  + 获取Data接口：getExternalFilesDirs

#### App-specific目录内部多媒体文件

App-specific目录内部多媒体文件：

* App自身访问，跟[1.3.3.1. App自身App-specific目录](#_自己内部sandbox)一样
* 其他App访问
  + 默认情况下Media Scanner不会扫描App-specific里面的多媒体文件，如果需要扫描需要通过MediaScannerConnection.scanFile添加到MediaProvider数据库中

访问方式跟[2.3.2. 读写公共目录](#_读写公共目录)一样。

* + App通过ContentProvider共享出去

#### 其他App目录文件

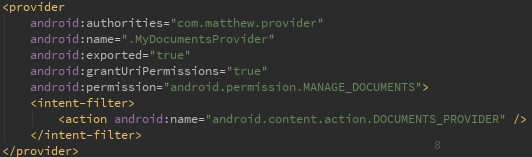
App是Filtered View，其他App无法直接访问当前App私有目录，需要通过下面方法：

##### 通过SAF文件

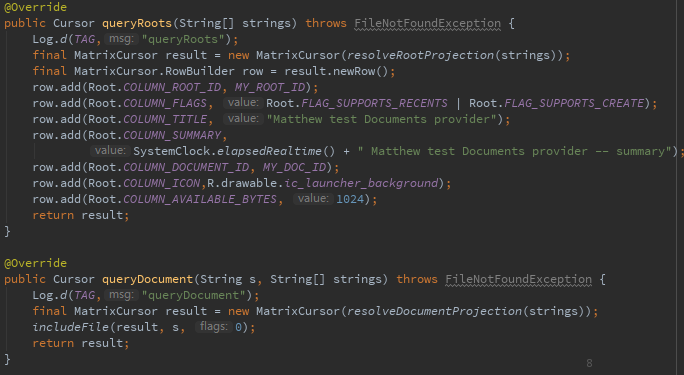
* 共享App自定义DocumentsProvider

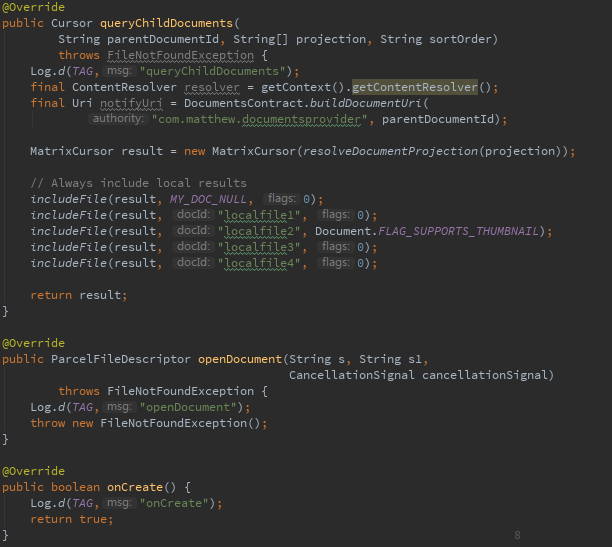
App自定义DocumentsProvider需要做以下步骤：

1. 指定DocumentsProvider

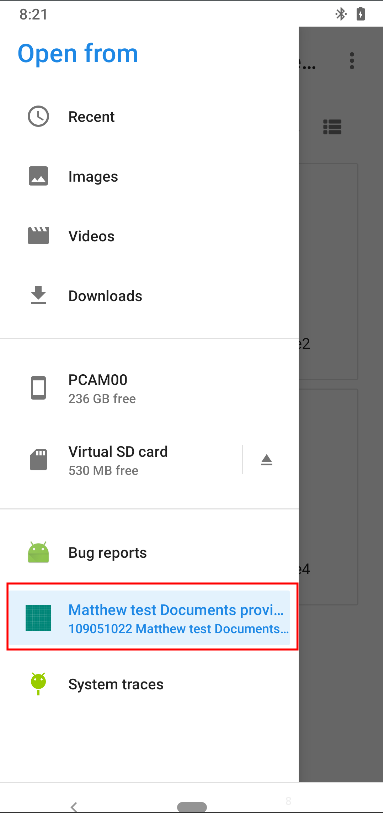
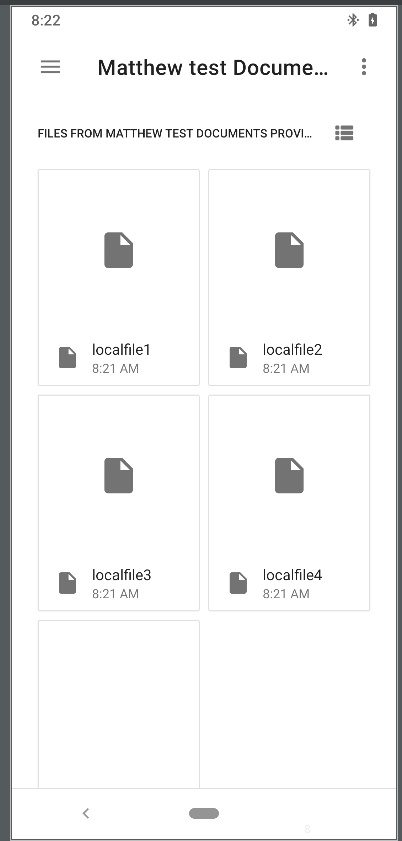


1. DocumentsProvider实现基本接口：





* 访问App通过ACTION\_OPEN\_DOCUMENT，启动浏览

##### 共享App实现FileProvider

FileProvider具体使用参考：

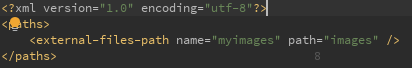
<https://developer.android.com/training/secure-file-sharing/setup-sharing>

这边总结一下大概步骤：

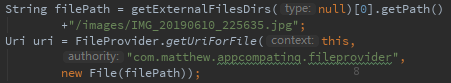
* 指定App FileProvider



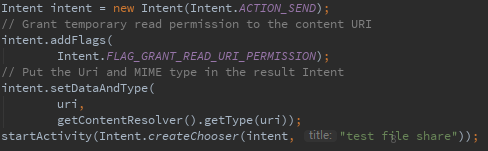
* 指定文件路径，配置文件必须要放到res/xml中



* 获取分享Uri



* 设置权限，并且发送Uri

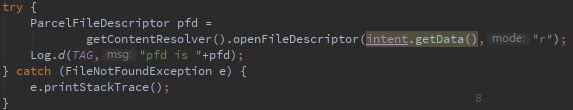




* 接收App，设置接受的inter-filter



* 接收并处理Uri



##### App自定义私有Provider

App可以实现自定义ContentProvider，尤其是内部文件共享，但是不希望UI交互。

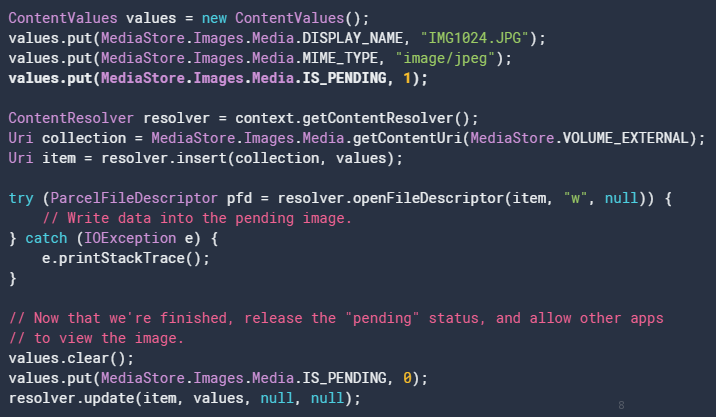
### MediaStore \_data字段

MediaStore中，DATA即（\_data）字段，在Android Q中开始废弃。读写文件需要通过openFileDescriptor。

### MediaStore文件Pending状态

Android Q上，MediaStore中添加了一个IS\_PENDING Flag，用于标记当前文件时Pending状态。

其他App通过MediaStore查询文件，如果没有设置setIncludePending接口，查询不到设置为Pending状态的文件，这就给App专享访问此文件。在一些情况下使用，例如在下载的时候：下载中，文件是Pending状态🡪下载完成，文件Pending状态置为0。



### MediaColumns.RELATIVE\_PATH设置存储路径

Android Q上，通过MediaStore存储到公共目录的文件，除了2.3.2.1.2节[Uri跟公共目录关系](#_URI跟公共目录关系)中规定的每一个存储空间的一级目录外，可以通过MediaColumns.RELATIVE\_PATH来指定存储的次级目录，这个目录可以使多级，具体代码如下：

* ContentResolver insert方法

通过values.put(Media.RELATIVE\_PATH,"Pictures/album/family ")指定存储目录。其中，Pictures是一级目录，album/family是子目录。

* ContentResolver update方法

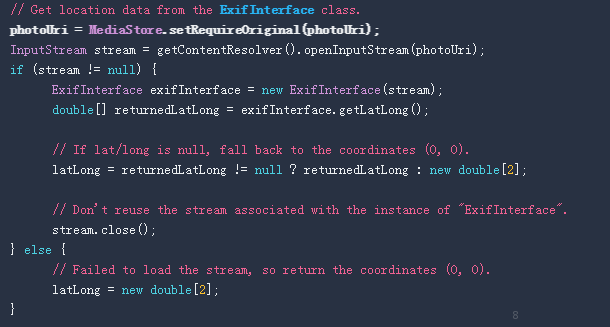
通过values.put(Media.RELATIVE\_PATH,"Pictures/album/family ")指定存储目录。通过update方法，可以移动存储地方。

### 访问图片Exif Metadata

Android Q上， App如果需要访问图片上的Exif Metadata，需要做下列事情：

* 申请ACCESS\_MEDIA\_LOCATION权限
* 通过MediaStore.setRequireOriginal返回新Uri

Demo Code如下：



### App Filtered View，访问权限总结

App访问不同目录的权限总结如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 文件位置 | 需要权限 | 访问方式 | App卸载是否保存 |
| App-specific目录 | 无 | getExternalFilesDir() | 不保留 |
| Media文件  (photos, videos, audio) | 访问其他app文件，需要  READ\_EXTERNAL\_STORAGE  修改其他app文件，需要  WRITE\_EXTERNAL\_STORAGE | MediaStore | 保留 |
| Downloads | 无 | SAF | 保留 |

### 应用卸载

如果App在AndroidManifest.xml中声明：android:hasFragileUserData="true"

卸载应用会有提示是否保留App数据：



### App数据迁移

Android Q上，App TargetSDK>=Q默认是Filtered View。App如果是Filtered View，会涉及到数据的迁移，不然会导致旧数据无法使用。可以从下面几方面着手数据迁移：

* App需要在Legacy View下才能拥有完整操作存储的权限
* App存放在非公共区域的文件，可以通过SAF访问

通过SAF选择目录文件，用户选择访问App文件。



* App可以将需要保存的文件：

Images、Video、Audio放到对应的公共目录，其他文件卸载后不删除文件可以放到Downloads下面。

### MediaStore Queries

在使用MediaStore进行query动作的时候，使用Projection时，Column Name要在MediaStore中定义好的。

# WRITE\_MEDIA\_STORAGE权限

## 背景

WRITE\_MEDIA\_STORAGE是一个很大强大的权限，能够允许App获取访问所有存储设备的权限。访问所有存储设备的权限，这个应当只赋予Media Stack。

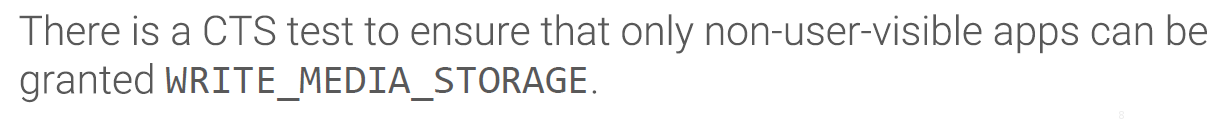
## 兼容影响

在Android系统中，规定了WRITE\_MEDIA\_STORAGE能够获取media\_rw用户组：



* 对于所有的可移动存储设备，例如T卡、U盘，在Mount到Android中的时候，对于普通App只有读权限，没有写权限，只有是media\_rw用户组App，才能够写可移动存储设备
* 对于Android Q上，Scoped Storage，可以通过此权限，设置App运行为兼容模式
* Android CTS会进行测试，User-launchable 的App，不能申请此权限

具体参考《Android Bootcamp 2019 - Privacy Overview.pdf》。



## 适配

* App如果需要访问Media或者外置存储设备，可以通过MediaStore或者Storage Access Framework（SAF）接口。