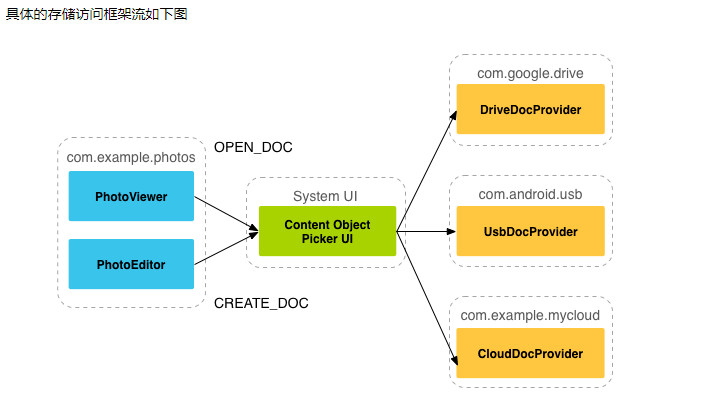
**存储访问框架 (SAF)**

**一． 概述**

Android 4.4（API 级别 19）引入了存储访问框架 (Storage Access Framework)。SAF 让用户能够在其所有首选文档存储提供程序中方便地浏览并打开文档、图像以及其他文件。 用户可以通过易用的标准 UI，以统一方式在所有应用和提供程序中浏览文件和访问最近使用的文件。



存储访问框架SAF 包括以下内容：

* **文档提供程序** ：ConentProvider的子类，允许存储服务显示其管理的文件。 文档提供程序作为 DocumentsProvider 类的子类实现。文档提供程序的架构基于传统文件层次结构。Android 平台包括若干内置文档提供程序，操作sd卡对应的为ExternalStorageProvider。
* **客户端应用** ：就是我们平时的app，它调用 ACTION\_OPEN\_DOCUMENT ,ACTION\_CREATE\_DOCUMENT ,ACTION\_OPEN\_DOCUMENT\_TREE这三种Intent的Action,来实现打开，创建文档，以及打开文档树。
* **选取器** ： 一种系统 UI，我们称为DocumentUi,允许用户访问所有满足客户端应用搜索条件的文档提供程序内的文档。这个DocumentUI无桌面图标和入口，只能通过上面的Intent访问。

在SAF框架中，我们的app应用和DocumentProvider之间并不产生直接的交互，而是通过**DocumentUi**进行。

**二．SAF框架的使用**

上文已经讲过，SAF框架的使用是通过DocumentUI的选择器来间接进行的，没法直接进行文件的操作。

使用方法如下：

1. 打开文件

private static final int READ\_REQUEST\_CODE = 42;

...

public void performFileSearch() {

Intent intent = new Intent(Intent.ACTION\_OPEN\_DOCUMENT);

//过滤器只显示可以打开的结果

intent.addCategory(Intent.CATEGORY\_OPENABLE);

//要搜索通过已安装的存储提供商提供的所有文档

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

startActivityForResult(intent, READ\_REQUEST\_CODE);

}

@Override

public void onActivityResult(int requestCode, int resultCode,Intent resultData) {

//使用resultdata.getdata ( )提取该URI

if (requestCode == READ\_REQUEST\_CODE && resultCode == Activity.RESULT\_OK) {

Uri uri = null;

if (resultData != null) {

uri = resultData.getData();

Log.i(TAG, "Uri: " + uri.toString());

showImage(uri);

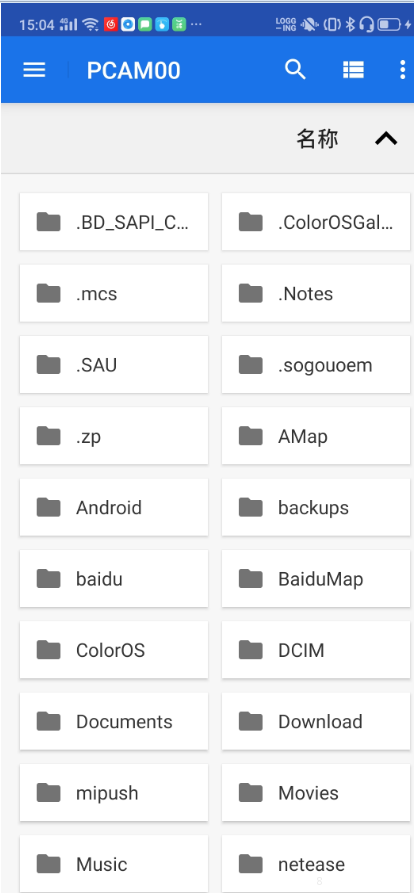
}

}

}

返回Uri：

content://com.android.externalstorage.documents/document/primary%3ADCIM%2FCamera%2FIMG20190607162534.jpg



1. **打开文件树**

Intent intent = new Intent(Intent.ACTION\_OPEN\_DOCUMENT\_TREE);

startActivityForResult(intent, OPEN\_TREE\_CODE);

private void handleTreeAction(Intent data){

Uri treeUri = data.getData();

//授予打开的文档树永久性的读写权限

final int takeFlags = intent.getFlags()

& (Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION

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

getContentResolver().takePersistableUriPermission(uri, takeFlags);

//使用DocumentFile构建一个根文档，之后的操作可以在该文档上进行

mRoot = DocumentFile.fromTreeUri(this, treeUri);

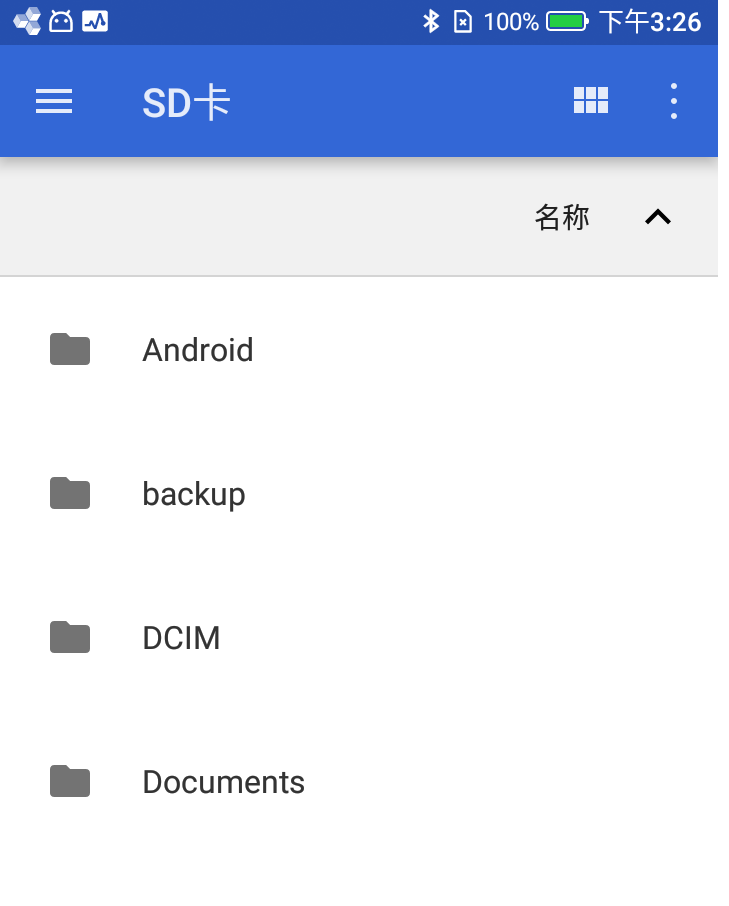
//显示结果toast

showToast(" open tree uri "+treeUri);

}

返回的Uri：

content://com.android.externalstorage.documents/tree/primary%3AColorOS



* 对于我们打开的文档树，系统会赋予我们对该文档树下所有文档的读写权限，因此我们可以自由的使用我们上面介绍的输入输出流或者文件的方式来进行读写，该授权会一直保留到用户重启设备。
* 但是有时候，我们需要能够永久性的访问这些文件的权限，而不是重启就需要重新授权，因此我们使用了takePersistableUriPermission方法来保留系统对我们的uri的授权，即使设备重启也不影响。
* 我们可能保存了应用最近访问的 URI，但它们可能不再有效 — 另一个应用可能已删除或修改了文档。 因此，应该调用 getContentResolver().takePersistableUriPermission() 以检查有无最新数据。
* 拿到了根目录的uri,我们就可用使用DocumentFile辅助类来方便的进行创建，删除文件等操作了。

1. **创建文件 ACTION\_CREATE\_DOCUMENT**

private void createDocument(){

Intent intent = new Intent(Intent.ACTION\_CREATE\_DOCUMENT);

//设置创建的文件是可打开的

intent.addCategory(Intent.CATEGORY\_OPENABLE);

//设置创建的文件的minitype为文本类型

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

//设置创建文件的名称，注意SAF中使用minitype而不是文件的后缀名来判断文件类型。

intent.putExtra(Intent.EXTRA\_TITLE, "123.txt");

startActivityForResult(intent,CREATE\_DOCUMENT\_CODE);

}

private void handleCreateDocumentAction(Intent data){

if (data == null) {

return;

}

BufferedWriter bw = null;

try {

OutputStream os = getContentResolver().openOutputStream(uri);

bw = new BufferedWriter(new OutputStreamWriter(os));

bw.write(" i am a text ");

showToast(" create document succeed uri "+uri);

} catch (IOException e) {

e.printStackTrace();

}finally {

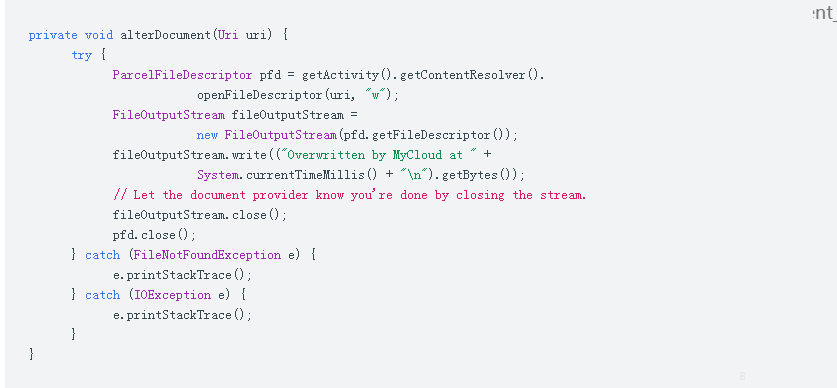
closeSafe(bw);

}

}

1. **编辑文档**

 在[onActivityResult()中获取到Uri之后，就可以对这个uri进行操作：](https://developer.android.com/reference/android/app/Activity.html" \l "onActivityResult(int,%20int,%20android.content.Intent))



1. **删除文档**

如果您获得了文档的 URI，并且文档的 [Document.COLUMN\_FLAGS](https://developer.android.com/reference/android/provider/DocumentsContract.Document.html#COLUMN_FLAGS) 包 [SUPPORTS\_DELETE](https://developer.android.com/reference/android/provider/DocumentsContract.Document.html#FLAG_SUPPORTS_DELETE)，便可以删除该文档。例如：

DocumentsContract.deleteDocument(getContentResolver(), uri);

1. **DocumentFile类的使用**

DocumentFile是google为了方便大家使用SAF进行文件操作，而推出的帮助类。它的api和java的File类比较接近，更符合一般用户的习惯，且内部实质都是使用了DocumentsContact类的方法来对文件进行操作。也就是说，我们也可以完全不使用DocumentFile而是使用DocumentsContact来完成SAF框架提供的文件操作，DocumentFile提供了三个静态工厂方法来创建自身。

fromSingleUri,该方法需要传入一个SAF返回的指向单个文件的uri,我们的ACTION\_OPEN\_DOCUMENT,ACTION\_CREATE\_DOCUMENT返回的uri就是该类型，其对应的实现类为SingleDocumentFile，代表的是单个的文件。

fromTreeUri，该方法传入指向文件夹的uri,我们的ACTION\_OPEN\_TREE返回的就是该类型，其对应的实现类为TreeDocumentFile，代表的是一个文件夹。

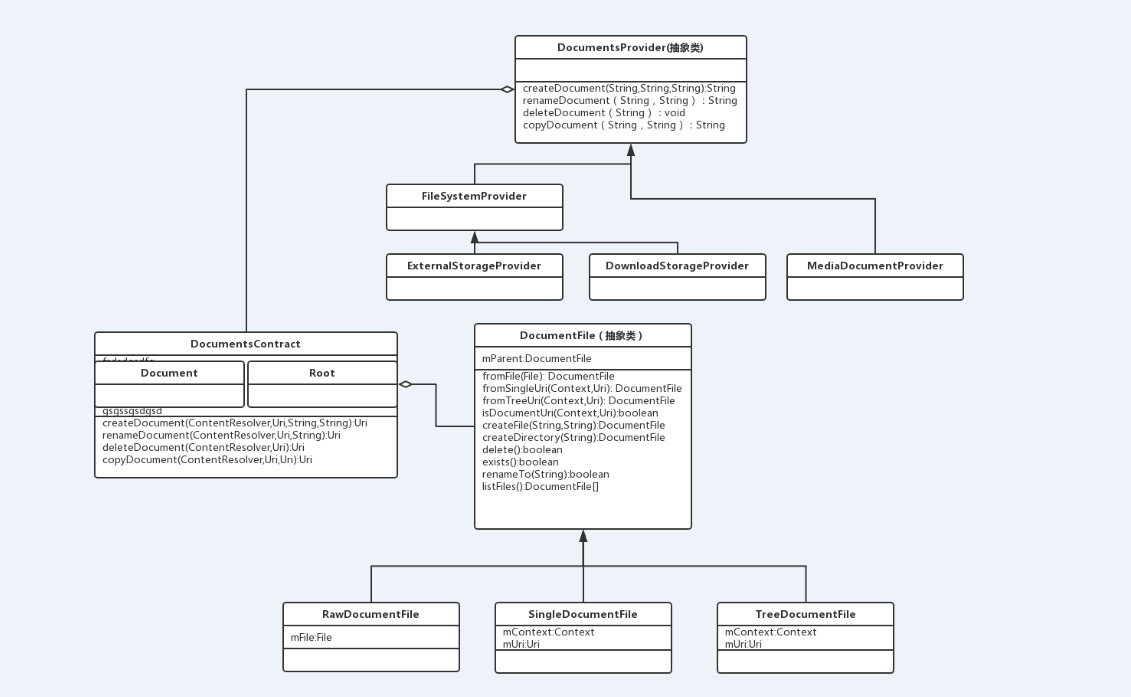
fromFile，该方法传入普通的File类，是对file类的一个模拟。

DocumentFile的方法总结如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名称** | **作用** | **SingleDocumentFile** | **TreeDocumentFile** |
| **isDocumentUri** | **判断uri类型是否为Document** | **有效** | **有效** |
| **createFile** | **创建文件** | **无效** | **有效** |
| **createDirectory** | **创建文件夹** | **无效** | **有效** |
| **isDocumentUri** | **判断uri类型是否为Document** | **有效** | **有效** |
| **isFile** | **判断是否为文件** | **有效** | **有效** |
| **isDirectory** | **判断是否为文件夹** | **有效** | **有效** |
| **canWrite** | **判断是否可写** | **有效** | **有效** |
| **canRead** | **判断是否可读** | **有效** | **有效** |
| **exists** | **判断文档是否存在** | **有效** | **有效** |
| **listFiles** | **列出该目录下所有文件** | **无效** | **有效** |
| **findFile** | **找出该目录下指定名称文件** | **无效** | **有效** |
| **createFile** | **创建文件** | **无效** | **有效** |
| **createDirectory** | **创建文件夹** | **无效** | **有效** |
| **delete** | **删除文档** | **有效** | **有效** |
| **renameTo** | **重命名文档** | **无效** | **有效** |

**三．SAF框架原理**

SAF框架的类关系图如下所示：



由类关系图可以看出，DocumentFile工具类最终是通过DocumentsContract来实现操作的，而DocumentsContract最终操作的Provider是DocumentsProvider。DocumentsProvider有三类。ExternalStorageProvider是外置SD卡对应的Provider，DownloadStorageProvider是下载对应的Provider。

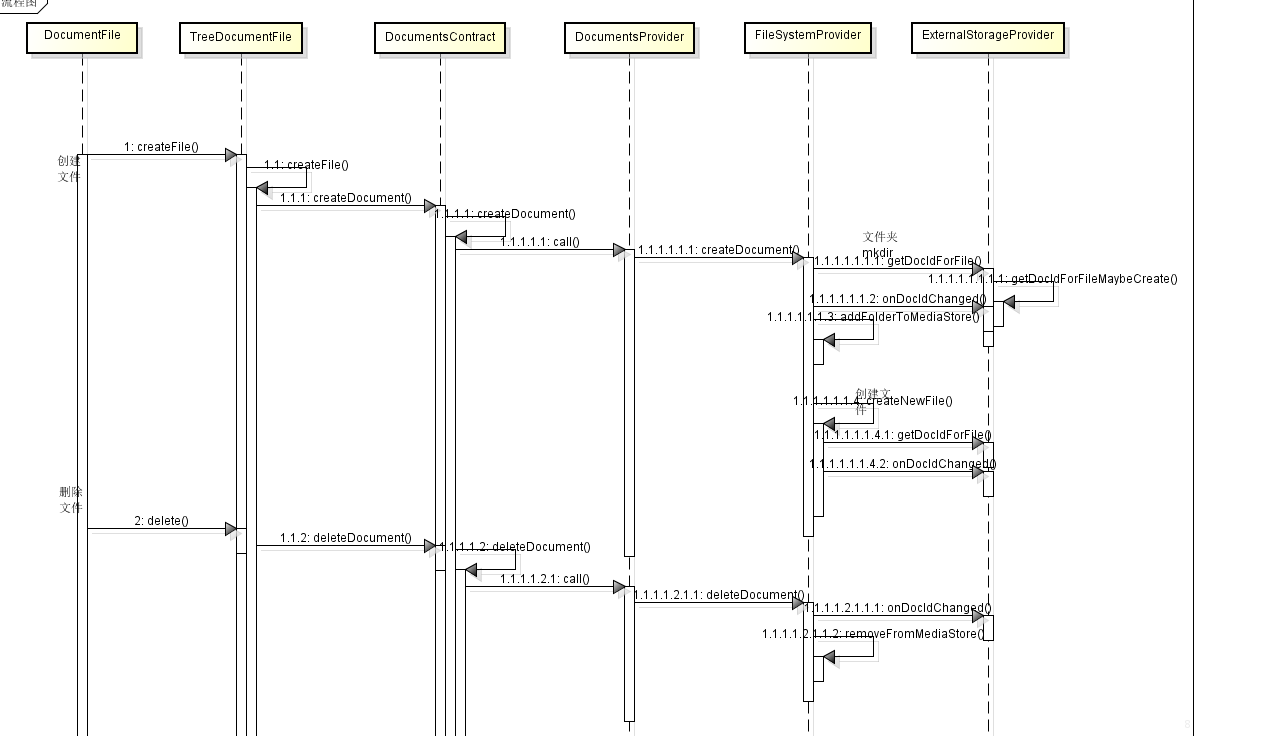
ExternalStorageProvider：com.android.externalstorage.documents

DownloadStorageProvider：com.android.providers.downloads.documents

MediaDocumentProvider：com.android.providers.media.documents

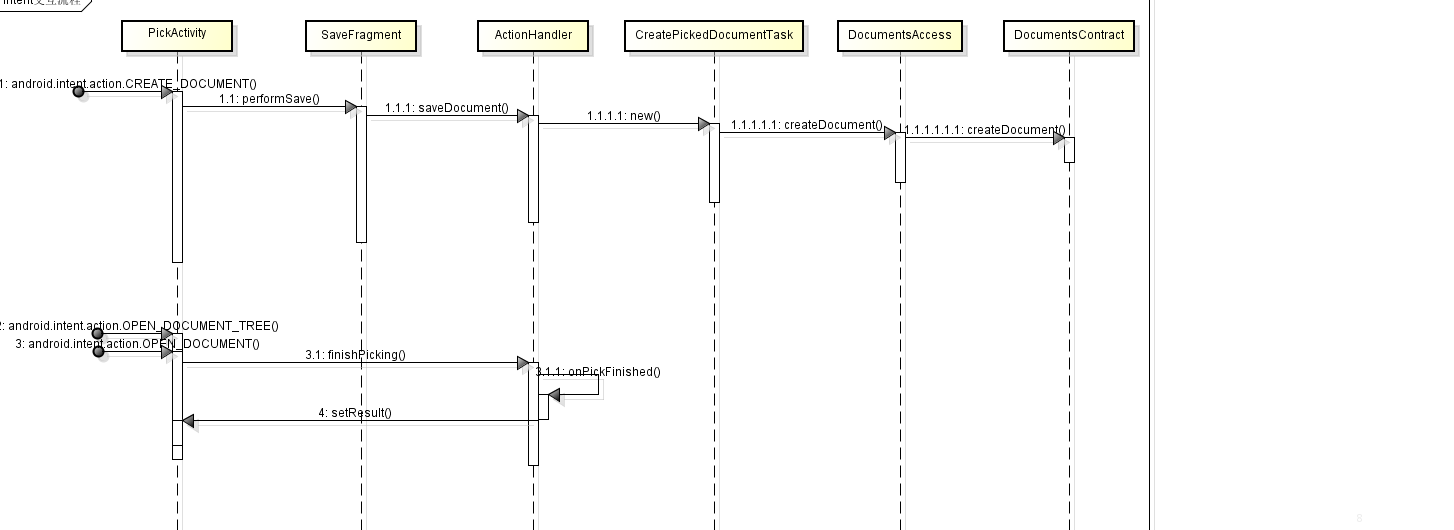
**下面具体分析下创建，修改，删除文件的流程**

可以看出DocumentFile辅助类最终也是通过DocumentsContract来操作DocumentsProvider



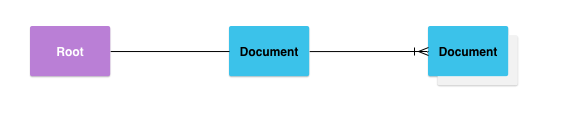
下面看下跳到选择PickerUI的流程：

PickerUI最终也调到了DocumentsContract中。



**2. DocumentProvider中的文档组织形式**

在文档提供程序内，数据结构采用传统的文件层次结构，如下图所示：



* 每个DocumentProvider都可能有1个或多个做为文档结构树的Root根目录，每个根目录都有唯一的COLUMN\_ROOT\_ID,并且指向该根目录下表示内容的文档。
* 每个根目录下都有一个文档，该文档指向1到n个文档，而其中的每个文档又可以指向1到N个文档，从而形成树形的文档结构。
* 每个Document都会有唯一的COLUMN\_DOCUMENT\_ID用以引用它们，文档id具有唯一性，并且一旦发放就不得更改，因为它们用于所有设备重启过程中的永久性 URI 授权。
* 文档可以是可打开的文件（具有特定 MIME 类型）或包含附加文档的目录（具有 MIME\_TYPE\_DIR MIME 类型）。
* 每个文档都可以具有不同的功能，如 COLUMN\_FLAGS 所述。例如，FLAG\_SUPPORTS\_WRITE、FLAG\_SUPPORTS\_DELETE 和 FLAG\_SUPPORTS\_THUMBNAIL。多个目录中可以包含相同的 COLUMN\_DOCUMENT\_ID。

Document:

|  |  |
| --- | --- |
| 字段名称 | 释义 |
| Document. COLUMN\_DOCUMENT\_ID | 文件唯一标识 |
| Document.COLUMN\_MIME\_TYPE | 文件类型 |
| Document.COLUMN\_DISPLAY\_NAME | 文件名称 |
| Document.COLUMN\_LAST\_MODIFIED | 最近修改时间 |
| Document.COLUMN\_FLAGS | 特性标志 |
| Document.COLUMN\_SUMMARY | 概述 |
| Document.COLUMN\_SIZE | 文件大小 |
| Document.COLUMN\_ICON | 图标 |

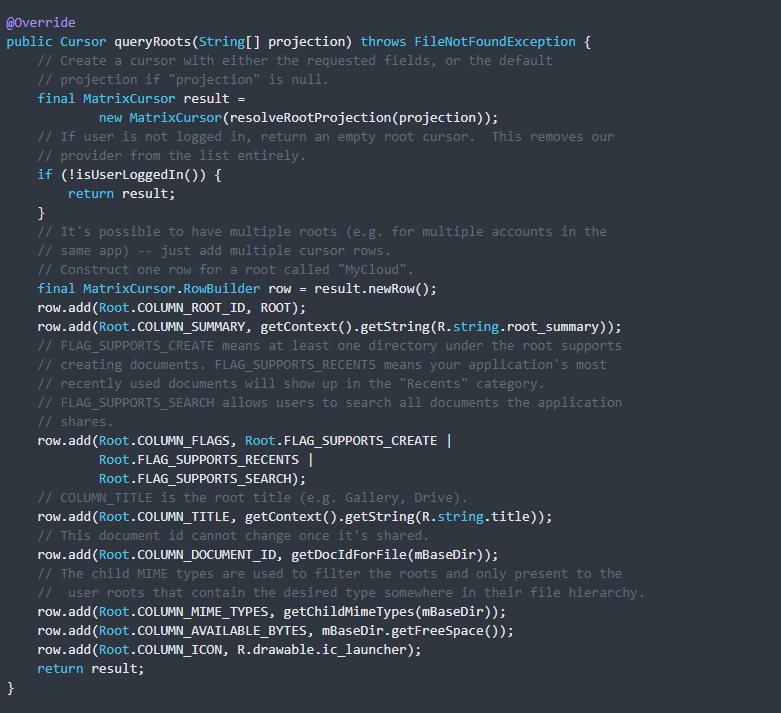
3．自定义DocumentProvider

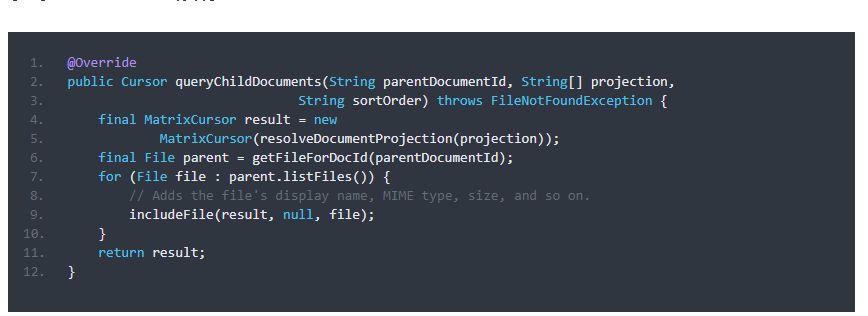
如果你希望自己应用的数据也能在documentsui中打开，你就需要写一个自己的document provider。（如果只是普通的文件操作，则不需要这么定义）

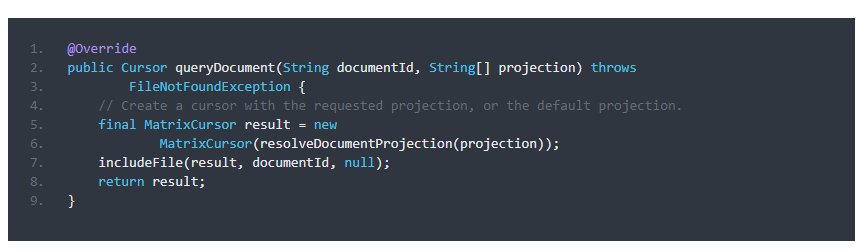
1. **首先需要在Manifest中声明自定义的provider：**



1. **实现DocumentProvider的基本接口**







查询文件详细信息使用

Query

**四．总结**

1. **SAF框架，并不是直接与与DocumentProvider直接打交道，而是通过DocumentUI来间接操作。**
2. **无论是通过Intent的方式，还是通过辅助类DocumentFile来进行文件操作，都需要获取uri，这个uri只能通过DocumentUI来返回，所以不是很方便。如果能接受通过DocumentUI来交互的，用SAF框架基本可以替代原有的文件操作方法**

补充问题：

1. 自定义DocumentsProvider需要实现以下四个方法，什么时候调用？

通过Uri匹配来调用

queryRoots()

[**mMatcher**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mMatcher).[addURI](http://opengrok.scm.adc.com:8080/source/s?defs=addURI&project=SDM710_P)([**mAuthority**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mAuthority), "root", [**MATCH\_ROOTS**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#MATCH_ROOTS));

[**mMatcher**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mMatcher).[addURI](http://opengrok.scm.adc.com:8080/source/s?defs=addURI&project=SDM710_P)([**mAuthority**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mAuthority), "root/\*", [**MATCH\_ROOT**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#MATCH_ROOT));

queryChildDocuments()

[**mMatcher**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mMatcher).[addURI](http://opengrok.scm.adc.com:8080/source/s?defs=addURI&project=SDM710_P)([**mAuthority**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mAuthority), "tree/\*/document/\*/children", [**MATCH\_CHILDREN\_TREE**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#MATCH_CHILDREN_TREE));

[**mMatcher**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mMatcher).[addURI](http://opengrok.scm.adc.com:8080/source/s?defs=addURI&project=SDM710_P)([**mAuthority**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mAuthority), "document/\*/children", [**MATCH\_CHILDREN**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#MATCH_CHILDREN));

queryDocument()

[**mMatcher**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mMatcher).[addURI](http://opengrok.scm.adc.com:8080/source/s?defs=addURI&project=SDM710_P)([**mAuthority**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#mAuthority), "document/\*", [**MATCH\_DOCUMENT**](http://opengrok.scm.adc.com:8080/source/xref/SDM710_P/android/frameworks/base/core/java/android/provider/DocumentsProvider.java#MATCH_DOCUMENT));

1. 三个provider的关系

ExternalStorageProvider：com.android.externalstorage.documents

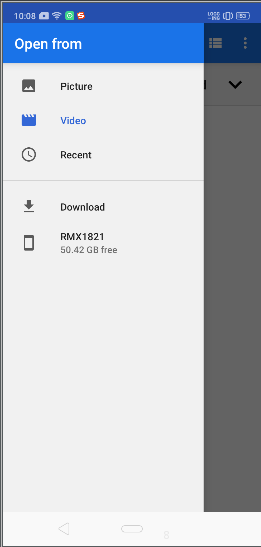
独立apk ：ExternalStorageProvider：操作外置SD卡

DownloadStorageProvider：com.android.providers.downloads.documents

DownloadProvier实现：

MediaDocumentProvider：com.android.providers.media.documents

MediaProvider实现

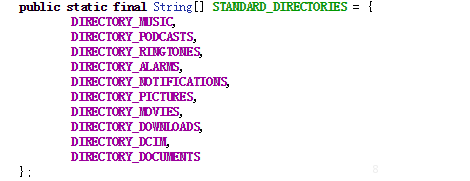


1. 授权的方式createintentAccess

StorageManager sm = (StorageManager) getSystemService(Context.*STORAGE\_SERVICE*);  
//获取存储器  
List<StorageVolume> list = sm.getStorageVolumes();  
Log.*d*("yinbin","list ==== "+ list.size());  
for(StorageVolume sv : list){  
 //遍历所有存储器，当它是Removable(包含外置sd卡，usb等)且已经装载时  
 if(TextUtils.*equals*(sv.getState(),Environment.*MEDIA\_MOUNTED*)) {  
 //调用StorageVolume的createAccessIntent方法  
 Intent i = sv.createAccessIntent(Environment.*DIRECTORY\_DOCUMENTS*);  
 if (i != null) {  
 startActivityForResult(i, *SDCARD\_AUTH\_CODE*);  
 return;  
 }  
  
 }  
}

只允许下面这几个目录申请，不允许传入null。

得到的uri同样可以使用SAF的方法进行文件操作。



1. ColorOS公共目录是如何生成的