

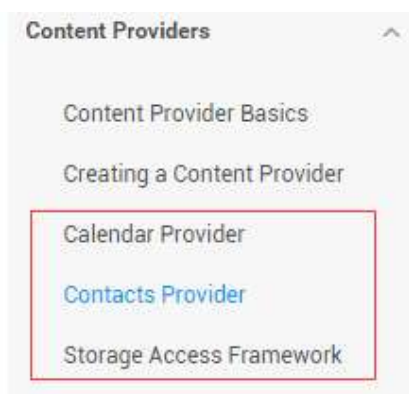
## 4.4.2 ContentProvider再探——Document Provider

分类 [Android 基础入门教程](#)

Android 基础入门教程(Q群号 : 153836263)

### 本节引言：

学完上一节，相信你已经知道如何去使用系统提供的ContentProvider或者自定义ContentProvider了，已经基本满足日常开发的需求了，有趣的是，我在官方文档上看到了另外这几个Provider：



**Calendar Provider**：日历提供者，就是针对针对日历相关事件的一个资源库，通过他提供的API，我们可以对日历，时间，会议，提醒等内容做一些增删改查！

**Contacts Provider**：联系人提供者，这个就不用说了，这个用得最多~后面有时间再回头翻译下这篇文章吧！

**Storage Access Framework(SAF)**：存储访问框架，4.4以后引入的一个新玩意，为用户浏览手机中的 存储内容提供了便利，可供访问的内容不仅包括：文档，图片，视频，音频，下载，而且包含所有由特定ContentProvider（须具有约定的API）提供的内容。不管这些内容来自于哪里，不管是哪个应用调用浏览系统文件内容的命令，系统都会用一个统一的界面让你去浏览。

其实就是一个内置的应用程序，叫做DocumentsUI，因为它的IntentFilter不带有LAUNCHER，所以我们并没有在桌面上找到这个东东！嘿嘿，试下下面的代码，这里我们选了两个手机来对比：分别是4.2的Lenovo S898T 和 5.0.1的Nexus 5做对比，执行下述代码：

```
Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
intent.addCategory(Intent.CATEGORY_OPENABLE);
intent.setType("image/*");
```

1.0 Android基础入门教程

1.0.1 2015年最新Android基...

1.1 背景相关与系统架构分析

1.2 开发环境搭建

1.2.1 使用Eclipse + ADT + S...

1.2.2 使用Android Studio开...

1.3 SDK更新不了问题解决

1.4 Genymotion模拟器安装

1.5.1 Git使用教程之本地仓...

1.5.2 Git之使用GitHub搭建...

1.6 .9(九妹)图片怎么玩

1.7 界面原型设计

1.8 工程相关解析(各种文件...

1.9 Android程序签名打包

1.11 反编译APK获取代码&...

2.1 View与ViewGroup的概念

2.2.1 LinearLayout(线性布局)

2.2.2 RelativeLayout(相对布...

2.2.3 TableLayout(表格布局)

2.2.4 FrameLayout(帧布局)

2.2.5 GridLayout(网格布局)

2.2.6 AbsoluteLayout(绝对...

2.3.1 TextView(文本框)详解

2.3.2 EditText(输入框)详解

2.3.3 Button(按钮)与ImageB...

2.3.4 ImageView(图像视图)

2.3.5.RadioButton(单选按钮...

2.3.6 开关按钮ToggleButton...

2.3.7 ProgressBar(进度条)

2.3.8 SeekBar(拖动条)

2.3.9 RatingBar(星级评分条)

2.4.1 ScrollView(滚动条)

```
startActivity(intent);
```

下面是运行结果：



2.4.2 Date & Time组件(上)

2.4.3 Date & Time组件(下)

2.4.4 Adapter基础讲解

2.4.5 ListView简单实用

2.4.6 BaseAdapter优化

2.4.7 ListView的焦点问题

2.4.8 ListView之checkbox错...

2.4.9 ListView的数据更新问题

2.5.0 构建一个可复用的自定...

2.5.1 ListView Item多布局的...

2.5.2 GridView(网格视图)的...

2.5.3 Spinner(列表选项框)...

2.5.4 AutoCompleteTextVie...

2.5.5 ExpandableListView(...

2.5.6 ViewPager(翻转视图)...

2.5.7 Toast(吐司)的基本使用

2.5.8 Notification(状态栏通...

2.5.9 AlertDialog(对话框)详解

2.6.0 其他几种常用对话框基...

2.6.1 PopupWindow(悬浮框...

2.6.2 菜单(Menu)

2.6.3 ViewPager的简单使用

2.6.4 DrawerLayout(官方侧...

3.1.1 基于监听的事件处理机制

3.2 基于回调的事件处理机制

3.3 Handler消息传递机制浅析

3.4 TouchListener PK OnTo...

3.5 监听EditText的内容变化

3.6 响应系统设置的事件(Co...

3.7 AsyncTask异步任务

3.8 Gestures(手势)

4.1.1 Activity初学乍练

4.1.2 Activity初窥门径

4.1.3 Activity登堂入室

4.2.1 Service初涉

4.2.2 Service进阶

4.2.3 Service精通

4.3.1 BroadcastReceiver牛...

4.3.2 BroadcastReceiver庖...

4.4.1 ContentProvider初探



右面这个就是4.4给我们带来的新玩意了，一般我们获取文件Url的时候就可以用到它~ 接下来简单的走下文档吧~

## 2.简单走下文档：

### 1 ) SAF框架的组成：

**Document provider**：一个特殊的ContentProvider，让一个存储服务(比如 Google Drive)可以 对外展示自己所管理的文件。它是**DocumentsProvider**的子类，另外，document-provider的存储格式 和传统的文件存储格式一致，至于你的内容如何存储，则完全决定于你自己，Android系统已经内置了几个这样的 Document provider，比如关于下载，图片以及视频的Document provider ！

**Client app**：一个普通的客户端软件，通过触发 **ACTION\_OPEN\_DOCUMENT** 和/或 **ACTION\_CREATE\_DOCUMENT** 就可以接收到来自于Document provider返回的内容，比如选择一个图片， 然后返回一个Uri。

**Picker**：类似于文件管理器的界面，而且是系统级的界面，提供额访问客户端过滤条件的 Document provider内容的通道，就是起说的那个 DocumentsUI程序！

4.4.2 ContentProvider再探...

4.5.1 Intent的基本使用

4.5.2 Intent之复杂数据的传递

5.1 Fragment基本概述

5.2.1 Fragment实例精讲—...

5.2.2 Fragment实例精讲—...

5.2.3 Fragment实例精讲—...

5.2.4 Fragment实例精讲—...

5.2.5 Fragment实例精讲—...

6.1 数据存储与访问之——文...

6.2 数据存储与访问之——S...

6.3.1 数据存储与访问之——...

6.3.2 数据存储与访问之——...

7.1.1 Android网络编程要学...

7.1.2 Android Http请求头与...

7.1.3 Android HTTP请求方...

7.1.4 Android HTTP请求方...

7.2.1 Android XML数据解析

7.2.2 Android JSON数据解析

7.3.1 Android 文件上传

7.3.2 Android 文件下载（1）

7.3.3 Android 文件下载（2）

7.4 Android 调用 Webservice

7.5.1 WebView(网页视图)基...

7.5.2 WebView和JavaScip...

7.5.3 Android 4.4后WebVie...

7.5.4 WebView文件下载

7.5.5 WebView缓存问题

7.5.6 WebView处理网页返...

7.6.1 Socket学习网络基础准备

7.6.2 基于TCP协议的Socket...

7.6.3 基于TCP协议的Socket...

7.6.4 基于UDP协议的Socke...

8.1.1 Android中的13种Draw...

8.1.2 Android中的13种Draw...

8.1.3 Android中的13种Draw...

8.2.1 Bitmap(位图)全解析 P...

8.2.2 Bitmap引起的OOM问题

8.3.1 三个绘图工具类详解

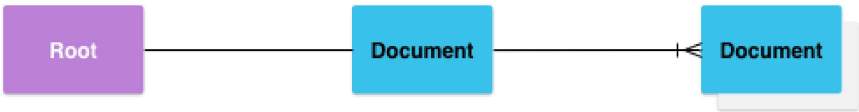
8.3.2 绘图类实战示例

一些特性：

- 用户可以浏览所有document provider提供的内容，而不仅仅是单一的应用程序
- 提供了长期、持续的访问document provider中文件的能力以及数据的持久化，用户可以实现添加、删除、编辑、保存document provider所维护的内容
- 支持多用户以及临时性的内容服务，比如USB storage providers只有当驱动安装成功才会出现

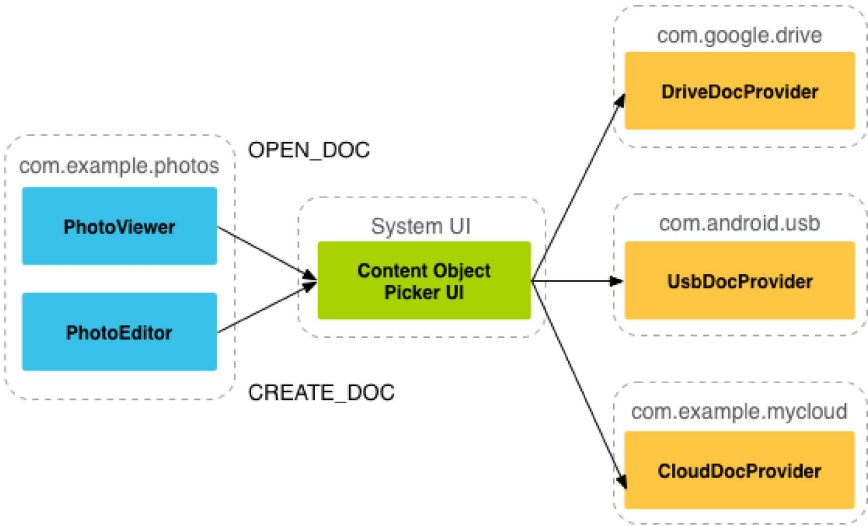
2 ) 概述：

SAF的核心是实现了DocumentsProvider的子类，还是一个ContentProvider。在一个document provider 中是以传统的文件目录树组织起来的：



3 ) 流程图：

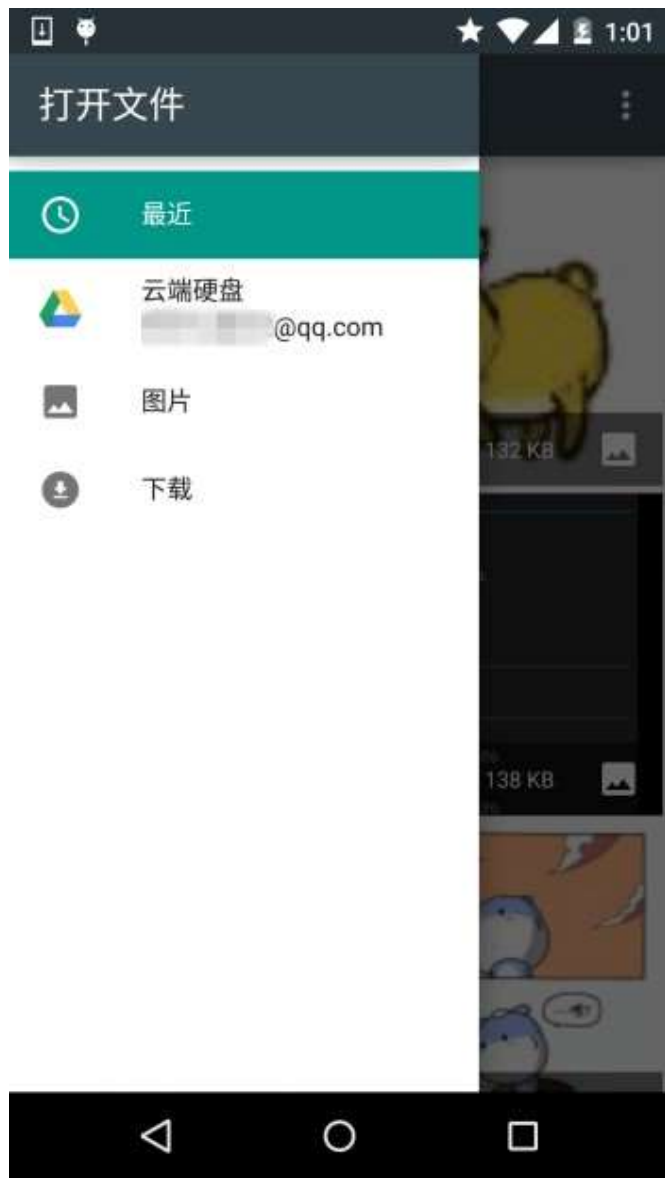
如上面所述，document provider data是基于传统的文件层次结构的，不过那只是对外的表现形式，如何存储你的数据，取决于你自己，只要你对海外的接口能够通过DocumentsProvider的api访问就可以。下面的流程图展示了一个photo应用使用SAF可能的结构：



分析：

从上图，我们可以看出Picker是链接调用者和内容提供者的一个桥梁！他提供并告诉调用者，可以选择 哪些内容提供者，比如这里的DriveDocProvider，UsbDocProvider，CloudDocProvider。当客户端触发了ACTION\_OPEN\_DOCUMENT或ACTION\_CREATE\_DOCUMENT的Intent，就会发生上述交互。当然我们还可以在Intent中增加过滤条件，比如限制MIME type的类型为"image"!

- 8.3.3 Paint API之—— Mask...
- 8.3.4 Paint API之—— Xferm...
- 8.3.5 Paint API之—— Xferm...
- 8.3.6 Paint API之—— Xferm...
- 8.3.7 Paint API之—— Xferm...
- 8.3.8 Paint API之—— Xferm...
- 8.3.9 Paint API之—— Color...
- 8.3.10 Paint API之—— Colo...
- 8.3.11 Paint API之—— Colo...
- 8.3.12 Paint API之—— Path...
- 8.3.13 Paint API之—— Sha...
- 8.3.14 Paint几个枚举/常量值...
- 8.3.15 Paint API之——Type...
- 8.3.16 Canvas API详解(Part 1)
- 8.3.17 Canvas API详解(Part...
- 8.3.18 Canvas API详解(Part...
- 8.4.1 Android动画合集之帧...
- 8.4.2 Android动画合集之补...
- 8.4.3 Android动画合集之属...
- 8.4.4 Android动画合集之属...
- 9.1 使用SoundPool播放音...
- 9.2 MediaPlayer播放音频与...
- 9.3 使用Camera拍照
- 9.4 使用MediaRecord录音
- 10.1 TelephonyManager(电...
- 10.2 SmsManager(短信管理...
- 10.3 AudioManager(音频管...
- 10.4 Vibrator(振动器)
- 10.5 AlarmManager(闹钟服务)
- 10.6 PowerManager(电源服...
- 10.7 WindowManager(窗口...
- 10.8 LayoutInflater(布局服务)
- 10.9 WallpaperManager(壁...
- 10.10 传感器专题(1)——相...
- 10.11 传感器专题(2)——方...
- 10.12 传感器专题(3)——加...
- 10.12 传感器专题(4)——其...
- 10.14 Android GPS初涉
- 11.0 《2015最新Android基...



就是上面这些东西，如果你还安装了其他看图的软件的话，也会在这里看到！

简单点说就是：客户端发送了上面两种Action的Intent后，会打开Picker UI，在这里会显示相关可用的 Document Provider，供用户选择，用户选后可以获得文件的相关信息！

#### 4) 客户端调用，并获取返回的Uri

实现代码如下：

```
public class MainActivity extends AppCompatActivity implements
View.OnClickListener {
    private static final int READ_REQUEST_CODE = 42;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn_show = (Button) findViewById(R.id.btn_show);
        btn_show.setOnClickListener(this);
    }

    @Override
```



反馈

```

public void onClick(View v) {
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT)
;
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    intent.setType("image/*");
    startActivityForResult(intent, READ_REQUEST_CODE);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == READ_REQUEST_CODE && resultCode == Activity.RESULT_OK) {
        Uri uri;
        if (data != null) {
            uri = data.getData();
            Log.e("HeHe", "Uri: " + uri.toString());
        }
    }
}
}

```

**运行结果：**比如我们选中那只狗，然后Picker UI自己会关掉，然后Logcat上可以看到这样一个uri:

```
com.jay.contentproviderdemo E/HeHe: Uri: content://com.android.providers.media.documents/document/image%3A73072
```

## 5) 根据uri获取文件参数

核心代码如下：

```

public void dumpImageMetaData(Uri uri) {
    Cursor cursor = getContentResolver()
        .query(uri, null, null, null, null, null);
    try {
        if (cursor != null && cursor.moveToFirst()) {
            String displayName = cursor.getString(
                cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME));
            Log.e("HeHe", "Display Name: " + displayName);
            int sizeIndex = cursor.getColumnIndex(OpenableColumns.SIZE);
            String size = null;
            if (!cursor.isNull(sizeIndex)) {
                size = cursor.getString(sizeIndex);
            } else {
                size = "Unknown";
            }
            Log.e("HeHe", "Size: " + size);
        }
    } finally {
        cursor.close();
    }
}

```



**运行结果：** 还是那只狗，调用方法后会输入文件名以及文件大小，以byte为单位

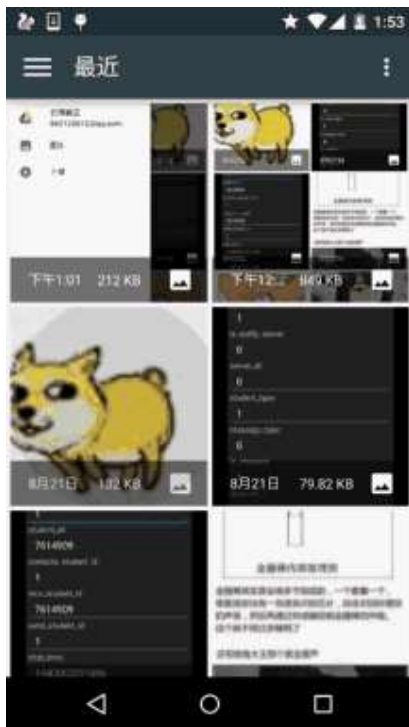
```
Uri: content://com.android.providers.media.documents/document/image%3A73072
Display Name: 779878443D7B744B454CD92631DC38F61C4DA3BB0.jpg
Size: 135338
```

## 6) 根据Uri获得Bitmap

核心代码如下：

```
private Bitmap getBitmapFromUri(Uri uri) throws IOException {
    ParcelFileDescriptor parcelFileDescriptor =
        getContentResolver().openFileDescriptor(uri, "r");
    FileDescriptor fileDescriptor = parcelFileDescriptor.ge
tFileDescriptor();
    Bitmap image = BitmapFactory.decodeFileDescriptor(fileD
escriptor);
    parcelFileDescriptor.close();
    return image;
}
```

**运行结果：**



## 7) 根据Uri获取输入流

核心代码如下：

```
private String readTextFromUri(Uri uri) throws IOException {
    InputStream inputStream = getContentResolver().openInputStr
eam(uri);
    BufferedReader reader = new BufferedReader(new InputStreamR
eader(
        inputStream));
    StringBuilder stringBuilder = new StringBuilder();
```

```
String line;
while ((line = reader.readLine()) != null) {
    stringBuilder.append(line);
}
fileInputStream.close();
parcelFileDescriptor.close();
return stringBuilder.toString();
}
```

上述的内容只告诉你通过一个Uri你可以知道什么，而Uri的获取则是通过SAF得到的！

## 8) 创建新文件以及删除文件：

**创建文件：**

```
private void createFile(String mimeType, String fileName) {
    Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    intent.setType(mimeType);
    intent.putExtra(Intent.EXTRA_TITLE, fileName);
    startActivityResult(intent, WRITE_REQUEST_CODE);
}
```

可在onActivityResult()中获取被创建文件的uri

**删除文件：**

前提是Document.COLUMN\_FLAGS包含**SUPPORTS\_DELETE**

```
DocumentsContract.deleteDocument(getContentResolver(), uri);
```

## 9) 编写一个自定义的Document Provider

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。下面介绍自定义DocumentsProvider的步骤：

API版本为19或者更高

在manifest.xml中注册该Provider

Provider的name为类名加包名，比如：

**com.example.android.storageprovider.MyCloudProvider**

Authority为包名+provider的类型名，如：

**com.example.android.storageprovider.documents**

**android:exported**属性的值为true

下面是Provider的例子写法：

```
<manifest... >
...
<uses-sdk
```



```
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />

        ....
        <provider
            android:name="com.example.android.storageprovider.M
yCloudProvider"
            android:authorities="com.example.android.storagepro
vider.documents"
            android:grantUriPermissions="true"
            android:exported="true"
            android:permission="android.permission.MANAGE_DOCUM
ENTS"

            android:enabled="@bool/atLeastKitKat">
            <intent-filter>
                <action android:name="android.content.action.DO
CUMENTS_PROVIDER" />
            </intent-filter>
        </provider>
    </application>

</manifest>
```

## 10 )DocumentsProvider的子类

至少实现如下几个方法：

```
queryRoots()

queryChildDocuments()

queryDocument()

openDocument()
```

还有些其他的方法，但并不是必须的。下面演示一个实现访问文件（file）系统的 DocumentsProvider的大致写法。

### Implement queryRoots

```
@Override
public Cursor queryRoots(String[] projection) throws FileNotFou
ndException {

    // Create a cursor with either the requested fields, or the
    default
    // projection if "projection" is null.
    final MatrixCursor result =
        new MatrixCursor(resolveRootProjection(projection))
    ;

    // If user is not logged in, return an empty root cursor.
    This removes our
    // provider from the list entirely.
    if (!isUserLoggedIn()) {
        return result;
```

```

    }

    // It's possible to have multiple roots (e.g. for multiple
accounts in the
    // same app) -- just add multiple cursor rows.
    // Construct one row for a root called "MyCloud".
    final MatrixCursor.RowBuilder row = result.newRow();
    row.add(Root.COLUMN_ROOT_ID, ROOT);
    row.add(Root.COLUMN_SUMMARY, getContext().getString(R.string
g.root_summary));

    // FLAG_SUPPORTS_CREATE means at least one directory under
the root supports
    // creating documents. FLAG_SUPPORTS_RECENTS means your app
lication's most
    // recently used documents will show up in the "Recents" ca
tegory.
    // FLAG_SUPPORTS_SEARCH allows users to search all document
s the application
    // shares.
    row.add(Root.COLUMN_FLAGS, Root.FLAG_SUPPORTS_CREATE |
            Root.FLAG_SUPPORTS_RECENTS |
            Root.FLAG_SUPPORTS_SEARCH);

    // COLUMN_TITLE is the root title (e.g. Gallery, Drive).
    row.add(Root.COLUMN_TITLE, getContext().getString(R.string
title));

    // This document id cannot change once it's shared.
    row.add(Root.COLUMN_DOCUMENT_ID, getDocIdForFile(mBaseDir))
;

    // The child MIME types are used to filter the roots and on
ly present to the
    // user roots that contain the desired type somewhere in t
heir file hierarchy.
    row.add(Root.COLUMN_MIME_TYPES, getChildMimeTypes(mBaseDir)
);
    row.add(Root.COLUMN_AVAILABLE_BYTES, mBaseDir.getFreeSpace(
));
    row.add(Root.COLUMN_ICON, R.drawable.ic_launcher);

    return result;
}

```

## Implement queryChildDocuments

```

public Cursor queryChildDocuments(String parentDocumentId, Stri
ng[] projection,
                                String sortOrder) throws FileNotF
oundException {

    final MatrixCursor result = new
        MatrixCursor(resolveDocumentProjection(projection))

```

```

;

    final File parent = getFileForDocId(parentDocumentId);
    for (File file : parent.listFiles()) {
        // Adds the file's display name, MIME type, size, and s
    on.

        includeFile(result, null, file);
    }
    return result;
}

```

## Implement queryDocument

```

@Override
public Cursor queryDocument(String documentId, String[] project
ion) throws
    FileNotFoundException {

    // Create a cursor with the requested projection, or the de
fault projection.
    final MatrixCursor result = new
        MatrixCursor(resolveDocumentProjection(projection))
;
    includeFile(result, documentId, null);
    return result;
}

```

好吧，文档中的内容大概就是这些了：一开始是想自己翻译的，后来在泡在网上的日子上找到了这一篇文档的中文翻译，就偷下懒了~

中文翻译链接：[android存储访问框架Storage Access Framework](#)

## 3.Android 4.4 获取资源路径问题：

其实这个SAF我们用得较多的地方无非是获取图片的Uri而已，而从上面的例子我们也发现了：我们这样获取的链接是这样的：

```
content://com.android.providers.media.documents/document/image%
3A69983
```

这样的链接，我们直接通过上面的方法获得uri即可！

当然，这个是4.4 或者以上版本的~！

如果是以前的版本：uri可能是这样的：

```
content://media/external/images/media/image%3A69983
```

这里贴下在别的地方看到的一个全面的方案，原文链接：[Android4.4中获取资源路径问题](#)

```

public static String getPath(final Context context, final Uri u
ri) {

```

```

        final boolean isKitKat = Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT;
        // DocumentProvider
        if (isKitKat && DocumentsContract.isDocumentUri(context, uri)) {
            // ExternalStorageProvider
            if (isExternalStorageDocument(uri)) {
                final String docId = DocumentsContract.getDocumentId(uri);

                final String[] split = docId.split(":");
                final String type = split[0];

                if ("primary".equalsIgnoreCase(type)) {
                    return Environment.getExternalStorageDirectory() + "/" + split[1];
                }

                // TODO handle non-primary volumes
            }
            // DownloadsProvider
            else if (isDownloadsDocument(uri)) {

                final String id = DocumentsContract.getDocumentId(uri);

                final Uri contentUri = ContentUris.withAppendedId(
                    Uri.parse("content://downloads/public_downloads"), Long.valueOf(id));

                return getDataColumn(context, contentUri, null, null);
            }
            // MediaProvider
            else if (isMediaDocument(uri)) {
                final String docId = DocumentsContract.getDocumentId(uri);

                final String[] split = docId.split(":");
                final String type = split[0];
                Uri contentUri = null;
                if ("image".equalsIgnoreCase(type)) {
                    contentUri = MediaStore.Images.Media.EXTERNAL_CONTENT_URI;
                } else if ("video".equalsIgnoreCase(type)) {
                    contentUri = MediaStore.Video.Media.EXTERNAL_CONTENT_URI;
                } else if ("audio".equalsIgnoreCase(type)) {
                    contentUri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
                }

                final String selection = "_id=?";
                final String[] selectionArgs = new String[] {
                    split[1]
                };

                return getDataColumn(context, contentUri, selection, selectionArgs);
            }
        }
    }

```

```

    }
    // MediaStore (and general)
    else if ("content".equalsIgnoreCase(uri.getScheme())) {
        return getDataColumn(context, uri, null, null);
    }
    // File
    else if ("file".equalsIgnoreCase(uri.getScheme())) {
        return uri.getPath();
    }
    return null;
}

/**
 * Get the value of the data column for this Uri. This is useful for
 * MediaStore Uris, and other file-based ContentProviders.
 *
 * @param context The context.
 * @param uri The Uri to query.
 * @param selection (Optional) Filter used in the query.
 * @param selectionArgs (Optional) Selection arguments used in the query.
 * @return The value of the _data column, which is typically a file path.
 */
public static String getDataColumn(Context context, Uri uri, String selection,
                                   String[] selectionArgs) {

    Cursor cursor = null;
    final String column = "_data";
    final String[] projection = {
        column
    };

    try {
        cursor = context.getContentResolver().query(uri, projection, selection, selectionArgs,
            null);
        if (cursor != null && cursor.moveToFirst()) {
            final int column_index = cursor.getColumnIndexOrThrow(column);
            return cursor.getString(column_index);
        }
    } finally {
        if (cursor != null)
            cursor.close();
    }
    return null;
}

/**
 * @param uri The Uri to check.
 * @return Whether the Uri authority is ExternalStorageProvider

```

```
.
*/
public static boolean isExternalStorageDocument(Uri uri) {
    return "com.android.externalstorage.documents".equals(uri.getAuthority());
}

/**
 * @param uri The Uri to check.
 * @return Whether the Uri authority is DownloadsProvider.
 */
public static boolean isDownloadsDocument(Uri uri) {
    return "com.android.providers.downloads.documents".equals(uri.getAuthority());
}

/**
 * @param uri The Uri to check.
 * @return Whether the Uri authority is MediaProvider.
 */
public static boolean isMediaDocument(Uri uri) {
    return "com.android.providers.media.documents".equals(uri.getAuthority());
}
```

本节小结：

好的，关于本节android存储访问框架SAF就到这里吧，没什么例子，后面用到再深入研究吧，知道下就好，4.4后获取文件路径就简单多了~



在线实例	字符集&工具	最新更新	站点信息
<div><div>· HTML 实例</div><div>· CSS 实例</div><div>· JavaScript 实例</div><div>· Ajax 实例</div><div>· jQuery 实例</div><div>· XML 实例</div><div>· Java 实例</div></div>	<div><div>· HTML 字符集设置</div><div>· HTML ASCII 字符集</div><div>· HTML ISO-8859-1</div><div>· HTML 实体符号</div><div>· HTML 拾色器</div><div>· JSON 格式化</div></div>	<div><div>· Swift 正式开源</div><div>· PHP 7 正式发布</div><div>· Shell 编程快速入门</div><div>· Shell 文件包含</div><div>· Shell 输入/输出...</div></div>	<div><div>· 意见反馈</div><div>· 免责声明</div><div>· 关于我们</div><div>· 文章归档</div></div>

工具

- Shell printf 命令
- Shell 基本运算符

关注微信



Copyright © 2013-2015 菜鸟教程 **runoob.com** All Rights Reserved. 备案号：闽ICP备15012807号-1