存储访问框架

本文内容 显示详细信息

关键类

- `DocumentsProvider`
- `DocumentsContract`

视频

- DevBytes：Android 4.4 存储访问框架：提供程序
- DevBytes：Android 4.4 存储访问框架：客户端

代码示例

- 存储提供程序
- StorageClient

另请参阅

- 内容提供程序基础知识

Android 4.4（API 级别 19）引入了存储访问框架 (SAF)。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。

云存储服务或本地存储服务可以通过实现封装其服务的 `DocumentsProvider` 参与此生态系统。只需几行代码，便可将需要访问提供程序文档的客户端应用与 SAF 集成。

SAF 包括以下内容：

- **文档提供程序** — 一种内容提供程序，允许存储服务（如 Google Drive）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，但其实际数据存储方式由您决定。Android 平台包括若干内置文档提供程序，如 Downloads、Images 和 Videos。
- **客户端应用** — 一种自定义应用，它调用 `ACTION_OPEN_DOCUMENT` 和/或 `ACTION_CREATE_DOCUMENT` Intent 并接收文档提供程序返回的文件；
- **选取器** — 一种系统 UI，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。

SAF 提供的部分功能如下：

- 允许用户浏览所有文档提供程序而不仅仅是单个应用中的内容；
- 让您的应用获得对文档提供程序所拥有文档的长期、持久性访问权限。 用户可以通过此访问权限添加、编辑、保存和删除提供程序上的文件；
- 支持多个用户帐户和临时根目录，如只有在插入驱动器后才会出现的 USB 存储提供程序。

概览

概览

SAF 围绕的内容提供程序是 `DocumentsProvider` 类的一个子类。在*文档提供程序*内，数据结构采用传统的文件层次结构：

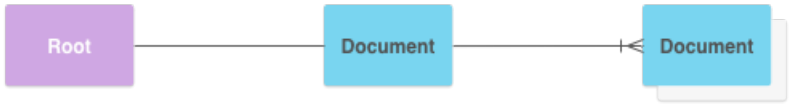

图 1. 文档提供程序数据模型。根目录指向单个文档，后者随即启动整个结构树的扇出。

请注意以下事项：

- 每个文档提供程序都会报告一个或多个作为探索文档结构树起点的“根目录”。每个根目录都有一个唯一的 `COLUMN_ROOT_ID`，并且指向表示该根目录下内容的文档（目录）。根目录采用动态设计，以支持多个帐户、临时 USB 存储设备或用户登录/注销等用例；
- 每个根目录下都有一个文档。该文档指向 1 至 *N* 个文档，而其中每个文档又可指向 1 至 *N* 个文档；
- 每个存储后端都会通过使用唯一的 `COLUMN_DOCUMENT_ID` 引用各个文件和目录来显示它们。文档 ID 必须具有唯一性，一旦发放便不得更改，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`。

控制流

如前文所述，文档提供程序数据模型基于传统文件层次结构。 不过，只要可以通过 `DocumentsProvider` API 访问数据，您实际上可以按照自己喜好的任何方式存储数据。例如，您可以使用基于标记的云存储来存储数据。

图 2 中的示例展示的是照片应用如何利用 SAF 访问存储的数据：

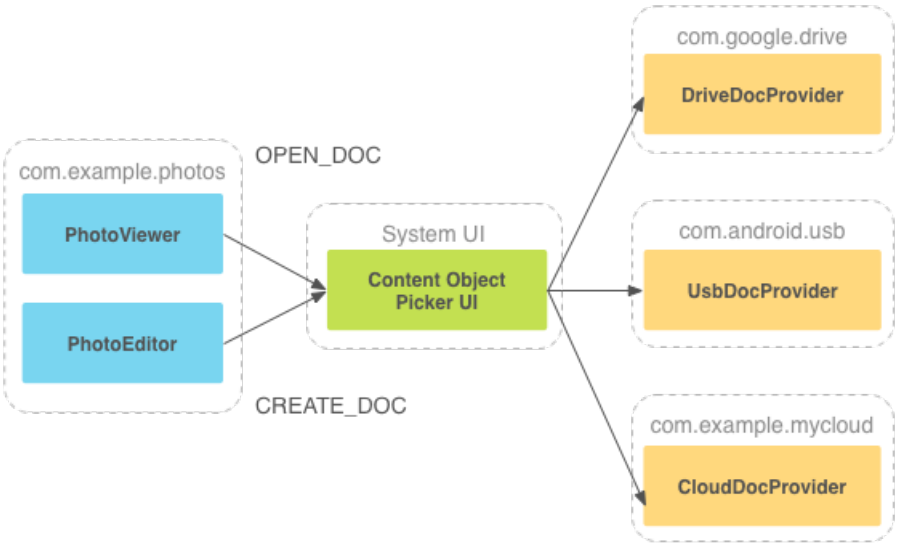

图 2. 存储访问框架流

请注意以下事项：

- 在 SAF 中，提供程序和客户端并不直接交互。 客户端请求与文件交互（即读取、编辑、创建或删除文件）的权限；
- 交互在应用（在本示例中为照片应用）触发 Intent `ACTION_OPEN_DOCUMENT` 或 `ACTION_CREATE_DOCUMENT` 后开始。Intent 可能包括进一步细化条件的过滤器 — 例如，“为我提供所有 MIME 类型为‘图像’的可打开文件”；
- Intent 触发后，系统选取器将检索每个已注册的提供程序，并向用户显示匹配的内容根目录；
- 选取器会为用户提供一个标准的文档访问界面，但底层文档提供程序可能与其差异很大。 例如，图 2 显示了一个 Google Drive 提供程序、一个 USB 提供程序和一个云提供程序。

图 3 显示了一个选取器，一位搜索图像的用户在其中选择了一个 Google Drive 帐户：

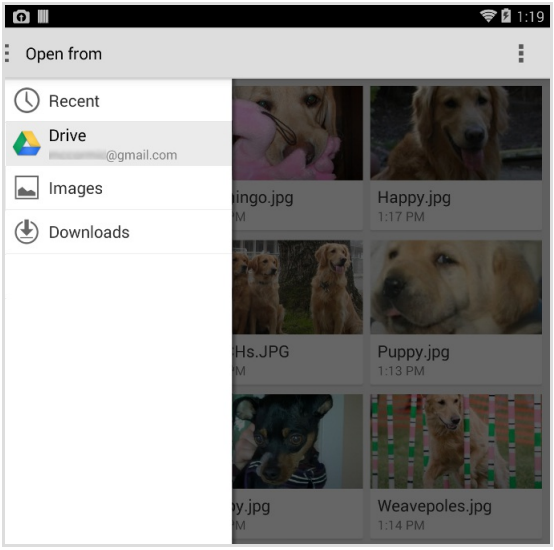

图 3. 选取器

当用户选择 Google Drive 时，系统会显示图像，如图 4 所示。从这时起，用户就可以通过提供程序和客户端应用支持的任何方式与它们进行交互。

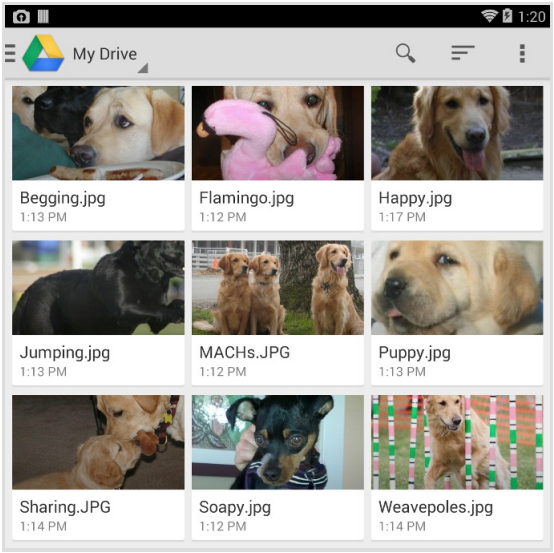

图 4. 图像

编写客户端应用

对于 Android 4.3 及更低版本，如果您想让应用从其他应用中检索文件，它必须调用 `ACTION_PICK` 或 `ACTION_GET_CONTENT` 等 Intent。然后，用户必须选择一个要从中选取文件的应用，并且所选应用必须提供一个用户界面，以便用户浏览和选取可用文件。

对于 Android 4.4 及更高版本，您还可以选择使用 `ACTION_OPEN_DOCUMENT` Intent，后者会显示一个由系统控制的选取器 UI，用户可以通过它浏览其他应用提供的所有文件。用户只需通过这一个 UI 便可从任何受支持的应用中选取文件。

`ACTION_OPEN_DOCUMENT` 并非设计用于替代 `ACTION_GET_CONTENT`。应使用的 Intent 取决于应用的需要：

- 如果您只想让应用读取/导入数据，请使用 `ACTION_GET_CONTENT`。使用此方法时，应用会导入数据（如图像文件）的副本；
- 如果您想让应用获得对文档提供程序所拥有文档的长期、持久性访问权限，请使用 `ACTION_OPEN_DOCUMENT`。 例如，允许用户编辑存储在文档提供程序中的图像的照片编辑应用。

本节描述如何编写基于 `ACTION_OPEN_DOCUMENT` 和 `ACTION_CREATE_DOCUMENT` Intent 的客户端应用。

搜索文档

以下代码段使用 `ACTION_OPEN_DOCUMENT` 来搜索包含图像文件的文档提供程序：

```
private static final int READ_REQUEST_CODE = 42;
...
/**
 * Fires an intent to spin up the "file chooser" UI and select an image.
 */
public void performFileSearch() {

    // ACTION_OPEN_DOCUMENT is the intent to choose a file via the system's file
    // browser.
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);

    // Filter to only show results that can be "opened", such as a
    // file (as opposed to a list of contacts or timezones)
    intent.addCategory(Intent.CATEGORY_OPENABLE);

    // Filter to show only images, using the image MIME data type.
    // If one wanted to search for ogg vorbis files, the type would be "audio/ogg".
    // To search for all documents available via installed storage providers,
    // it would be "*/*".
    intent.setType("image/*");

    startActivityForResult(intent, READ_REQUEST_CODE);
}
```

请注意以下事项：

- 当应用触发 `ACTION_OPEN_DOCUMENT` Intent 时，后者会启动一个选取器来显示所有匹配的文档提供程序
- 在 Intent 中添加类别 `CATEGORY_OPENABLE` 可对结果进行过滤，以仅显示可以打开的文档（如图像文件）
- 语句 `intent.setType("image/*")` 可做进一步过滤，以仅显示 MIME 数据类型为图像的文档

处理结果

用户在选取器中选择文档后，系统就会调用 `onActivityResult()`。指向所选文档的 URI 包含在 `resultData` 参数中。使用 `getData()` 提取 URI。获得 URI 后，即可使用它来检索用户想要的文档。例如：

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

    // The ACTION_OPEN_DOCUMENT intent was sent with the request code
    // READ_REQUEST_CODE. If the request code seen here doesn't match, it's the
    // response to some other intent, and the code below shouldn't run at all.

    if (requestCode == READ_REQUEST_CODE && resultCode == Activity.RESULT_OK) {
        // The document selected by the user won't be returned in the intent.
        // Instead, a URI to that document will be contained in the return intent
        // provided to this method as a parameter.
        // Pull that URI using resultData.getData().
        Uri uri = null;
        if (resultData != null) {
            uri = resultData.getData();
            Log.i(TAG, "Uri: " + uri.toString());
            showImage(uri);
        }
    }
}
```

检查文档元数据

获得文档的 URI 后，即可获得对其元数据的访问权限。以下代码段用于获取 URI 所指定文档的元数据并将其记入日志：

```
public void dumpImageMetaData(Uri uri) {

    // The query, since it only applies to a single document, will only return
    // one row. There's no need to filter, sort, or select fields, since we want
    // all fields for one document.
    Cursor cursor = getActivity().getContentResolver()
            .query(uri, null, null, null, null, null);

    try {
    // moveToFirst() returns false if the cursor has 0 rows.  Very handy for
    // "if there's anything to look at, look at it" conditionals.
        if (cursor != null && cursor.moveToFirst()) {

            // Note it's called "Display Name".  This is
            // provider-specific, and might not necessarily be the file name.
            String displayName = cursor.getString(
                    cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME));
            Log.i(TAG, "Display Name: " + displayName);

            int sizeIndex = cursor.getColumnIndex(OpenableColumns.SIZE);
            // If the size is unknown, the value stored is null.  But since an
            // int can't be null in Java, the behavior is implementation-specific,
            // which is just a fancy term for "unpredictable".  So as
            // a rule, check if it's null before assigning to an int.  This will
            // happen often:  The storage API allows for remote files, whose
            // size might not be locally known.
            String size = null;
            if (!cursor.isNull(sizeIndex)) {
                // Technically the column stores an int, but cursor.getString()
                // will do the conversion automatically.
                size = cursor.getString(sizeIndex);
            } else {
                size = "Unknown";
            }
            Log.i(TAG, "Size: " + size);
        }
    } finally {
        cursor.close();
    }
}
```

打开文档

获得文档的 URI 后，即可打开文档或对其执行任何其他您想要执行的操作。

位图

以下示例展示了如何打开 `Bitmap`：

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

请注意，您不应在 UI 线程上执行此操作。请使用 `AsyncTask` 在后台执行此操作。打开位图后，即可在 `ImageView` 中显示它。

获取 InputStream

以下示例展示了如何从 URI 中获取 `InputStream`。在此代码段中，系统将文件行读取到一个字符串中：

```
private String readTextFromUri(Uri uri) throws IOException {
    InputStream inputStream = getContentResolver().openInputStream(uri);
    BufferedReader reader = new BufferedReader(new InputStreamReader(
            inputStream));
    StringBuilder stringBuilder = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        stringBuilder.append(line);
    }
    fileInputStream.close();
    parcelFileDescriptor.close();
    return stringBuilder.toString();
}
```

创建新文档

您的应用可以使用 `ACTION_CREATE_DOCUMENT` Intent 在文档提供程序中创建新文档。要想创建文件，请为您的 Intent 提供一个 MIME 类型和文件名，然后通过唯一的请求代码启动它。系统会为您执行其余操作：

```
// Here are some examples of how you might call this method.
// The first parameter is the MIME type, and the second parameter is the name
// of the file you are creating:
//
// createFile("text/plain", "foobar.txt");
// createFile("image/png", "mypicture.png");

// Unique request code.
private static final int WRITE_REQUEST_CODE = 43;
...
private void createFile(String mimeType, String fileName) {
    Intent intent = new Intent(Intent.ACTION_CREATE_DOCUMENT);

    // Filter to only show results that can be "opened", such as
    // a file (as opposed to a list of contacts or timezones).
    intent.addCategory(Intent.CATEGORY_OPENABLE);

    // Create a file with the requested MIME type.
    intent.setType(mimeType);
    intent.putExtra(Intent.EXTRA_TITLE, fileName);
    startActivityForResult(intent, WRITE_REQUEST_CODE);
}
```

创建新文档后，即可在 `onActivityResult()` 中获取其 URI，以便继续向其写入内容。

删除文档

如果您获得了文档的 URI，并且文档的 `Document.COLUMN_FLAGS` 包含 `SUPPORTS_DELETE`，便可以删除该文档。例如：

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

编辑文档

您可以使用 SAF 就地编辑文本文档。以下代码段会触发 `ACTION_OPEN_DOCUMENT` Intent 并使用类别 `CATEGORY_OPENABLE` 以仅显示可以打开的文档。它会进一步过滤以仅显示文本文件：

```
private static final int EDIT_REQUEST_CODE = 44;
/**
 * Open a file for writing and append some text to it.
 */
 private void editDocument() {
    // ACTION_OPEN_DOCUMENT is the intent to choose a file via the system's
    // file browser.
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);

    // Filter to only show results that can be "opened", such as a
    // file (as opposed to a list of contacts or timezones).
    intent.addCategory(Intent.CATEGORY_OPENABLE);

    // Filter to show only text files.
    intent.setType("text/plain");

    startActivityForResult(intent, EDIT_REQUEST_CODE);
}
```

接下来，您可以从 `onActivityResult()`（请参阅**处理结果**）调用代码以执行编辑。以下代码段可从 `ContentResolver` 获取 `FileOutputStream`。默认情况下，它使用“写入”模式。最佳做法是请求获得所需的最低限度访问权限，因此如果您只需要写入权限，就不要请求获得读取/写入权限。

```
private void alterDocument(Uri uri) {
    try {
        ParcelFileDescriptor pfd = getActivity().getContentResolver().
                openFileDescriptor(uri, "w");
        FileOutputStream fileOutputStream =
                new FileOutputStream(pfd.getFileDescriptor());
        fileOutputStream.write(("Overwritten by MyCloud at " +
                System.currentTimeMillis() + "\n").getBytes());
        // Let the document provider know you're done by closing the stream.
        fileOutputStream.close();
        pfd.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

保留权限

当您的应用打开文件进行读取或写入时，系统会为您的应用提供针对该文件的 URI 授权。 该授权将一直持续到用户设备重启时。但假定您的应用是图像编辑应用，而且您希望用户能够直接从应用中访问他们编辑的最后 5 张图像。 如果用户的设备已经重启，您就需要将用户转回系统选取器以查找这些文件，这显然不是理想的做法。

为防止出现这种情况，您可以保留系统为您的应用授予的权限。 您的应用实际上是“获取”了系统提供的持久 URI 授权。 这使用户能够通过您的应用持续访问文件，即使设备已重启也不受影响：

```
final int takeFlags = intent.getFlags()
            & (Intent.FLAG_GRANT_READ_URI_PERMISSION
            | Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
// Check for the freshest data.
getContentResolver().takePersistableUriPermission(uri, takeFlags);
```

还有最后一个步骤。您可能已经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，您应该始终调用 `getContentResolver().takePersistableUriPermission()` 以检查有无最新数据。

编写自定义文档提供程序

如果您要开发为文件提供存储服务（如云保存服务）的应用，可以通过编写自定义文档提供程序，通过 SAF 提供您的文件。 本节描述如何执行此操作。

清单文件

要想实现自定义文档提供程序，请将以下内容添加到您的应用的清单文件：

- 一个 API 级别 19 或更高级别的目标；
- 一个声明自定义存储提供程序的 `<provider>` 元素；
- 提供程序的名称（即其类名），包括软件包名称。例如：`com.example.android.storageprovider.MyCloudProvider`；
- 授权的名称，即您的软件包名称（在本例中为 `com.example.android.storageprovider`）加内容提供程序的类型 (`documents`)。例如，`com.example.android.storageprovider.documents`。
- 属性 `android:exported` 设置为 `"true"`。您必须导出提供程序，以便其他应用可以看到；
- 属性 `android:grantUriPermissions` 设置为 `"true"`。此设置允许系统向其他应用授予对提供程序中内容的访问权限。 如需查看有关保留对特定文档授权的阐述，请参阅保留权限；
- `MANAGE_DOCUMENTS` 权限。默认情况下，提供程序对所有人可用。 添加此权限将限定您的提供程序只能供系统使用。此限制具有重要的安全意义；
- `android:enabled` 属性设置为在资源文件中定义的一个布尔值。此属性的用途是，在运行 Android 4.3 或更低版本的设备上停用提供程序。例如，`android:enabled="@bool/atLeastKitKat"`。 除了在清单文件中加入此属性外，您还需要执行以下操作；
  - 在 `res/values/` 下的 `bool.xml` 资源文件中，添加以下行：

    ```
    <bool name="atLeastKitKat">false</bool>
    ```

  - 在 `res/values-v19/` 下的 `bool.xml` 资源文件中，添加以下行：

    ```
    <bool name="atLeastKitKat">true</bool>
    ```

- 一个包括 `android.content.action.DOCUMENTS_PROVIDER` 操作的 Intent 过滤器，以便在系统搜索提供程序时让您的提供程序出现在选取器中。

以下是从一个包括提供程序的示例清单文件中摘录的内容：

```
<manifest... >
    ...
    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="19" />
        ....
        <provider
            android:name="com.example.android.storageprovider.MyCloudProvider"
            android:authorities="com.example.android.storageprovider.documents"
            android:grantUriPermissions="true"
            android:exported="true"
            android:permission="android.permission.MANAGE_DOCUMENTS"
            android:enabled="@bool/atLeastKitKat">
            <intent-filter>
                <action android:name="android.content.action.DOCUMENTS_PROVIDER" />
            </intent-filter>
        </provider>
    </application>

</manifest>
```

支持运行 Android 4.3 及更低版本的设备

`ACTION_OPEN_DOCUMENT` Intent 仅可用于运行 Android 4.4 及更高版本的设备。如果您想让应用支持 `ACTION_GET_CONTENT` 以适应运行 Android 4.3 及更低版本的设备，则应在您的清单文件中为运行 Android 4.4 或更高版本的设备停用 `ACTION_GET_CONTENT` Intent 过滤器。 应将文档提供程序和 `ACTION_GET_CONTENT` 视为具有互斥性。如果您同时支持这两者，您的应用将在系统选取器 UI 中出现两次，提供两种不同的方式来访问您存储的数据。 这会给用户造成困惑。

建议按照以下步骤为运行 Android 4.4 版或更高版本的设备停用 `ACTION_GET_CONTENT` Intent 过滤器：

1. 在 `res/values/` 下的 `bool.xml` 资源文件中，添加以下行：

```
<bool name="atMostJellyBeanMR2">true</bool>
```

2. 在 `res/values-v19/` 下的 `bool.xml` 资源文件中，添加以下行：

```
<bool name="atMostJellyBeanMR2">false</bool>
```

3. 添加一个 Activity 别名，为 4.4 版（API 级别 19）或更高版本停用 `ACTION_GET_CONTENT` Intent 过滤器。 例如：

```
<!-- This activity alias is added so that GET_CONTENT intent-filter
     can be disabled for builds on API level 19 and higher. -->
<activity-alias android:name="com.android.example.app.MyPicker"
        android:targetActivity="com.android.example.app.MyActivity"
        ...
        android:enabled="@bool/atMostJellyBeanMR2">
    <intent-filter>
        <action android:name="android.intent.action.GET_CONTENT" />
        <category android:name="android.intent.category.OPENABLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
        <data android:mimeType="video/*" />
    </intent-filter>
</activity-alias>
```

协定类

通常，当您编写自定义内容提供程序时，其中一项任务是实现协定类，如内容提供程序开发者指南中所述。 协定类是一种 `public final` 类，它包含对 URI、列名称、MIME 类型以及其他与提供程序有关的元数据的常量定义。 SAF 会为您提供这些协定类，因此您无需自行编写：

- `DocumentsContract.Document`
- `DocumentsContract.Root`

例如，当系统在您的文档提供程序中查询文档或根目录时，您可能会在游标中返回以下列：

```
private static final String[] DEFAULT_ROOT_PROJECTION =
        new String[]{Root.COLUMN_ROOT_ID, Root.COLUMN_MIME_TYPES,
        Root.COLUMN_FLAGS, Root.COLUMN_ICON, Root.COLUMN_TITLE,
        Root.COLUMN_SUMMARY, Root.COLUMN_DOCUMENT_ID,
        Root.COLUMN_AVAILABLE_BYTES,};
private static final String[] DEFAULT_DOCUMENT_PROJECTION = new
        String[]{Document.COLUMN_DOCUMENT_ID, Document.COLUMN_MIME_TYPE,
        Document.COLUMN_DISPLAY_NAME, Document.COLUMN_LAST_MODIFIED,
        Document.COLUMN_FLAGS, Document.COLUMN_SIZE,};
```

为 DocumentsProvider 创建子类

编写自定义文档提供程序的下一步是为抽象类 `DocumentsProvider` 创建子类。您至少需要实现以下方法：

- `queryRoots()`
- `queryChildDocuments()`
- `queryDocument()`
- `openDocument()`

这些只是您需要严格实现的方法，但您可能还想实现许多其他方法。 详情请参阅 `DocumentsProvider`。

实现 queryRoots

您实现的 `queryRoots()` 必须使用在 `DocumentsContract.Root` 中定义的列返回一个指向文档提供程序所有根目录的 `Cursor`。

在以下代码段中，`projection` 参数表示调用方想要返回的特定字段。 代码段会创建一个新游标，并为其添加一行 — 一个根目录，如

Downloads 或 Images 等顶层目录。 大多数提供程序只有一个根目录。有时您可能有多个根目录，例如，当您具有多个用户帐户时。 在这种情况下，只需再为游标添加一行。

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

    // 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.root_summary));

    // FLAG_SUPPORTS_CREATE means at least one directory under the root supports
    // creating documents. FLAG_SUPPORTS_RECENTS means your application's most
    // recently used documents will show up in the "Recents" category.
    // FLAG_SUPPORTS_SEARCH allows users to search all documents 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 only present to the
    //  user roots that contain the desired type somewhere in their 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;
}
```

实现 queryChildDocuments

您实现的 `queryChildDocuments()` 必须使用在 `DocumentsContract.Document` 中定义的列返回一个指向指定目录中所有文件的 `Cursor`。

当您在选取器 UI 中选择应用时，系统会调用此方法。它会获取根目录下某个目录内的子文档。可以在文件层次结构的任何级别调用此方法，并非只能从根目录调用。 以下代码段可创建一个包含所请求列的新游标，然后向游标添加父目录中每个直接子目录的相关信息。子目录可以是图像、另一个目录乃至任何文件：

```
@Override
public Cursor queryChildDocuments(String parentDocumentId, String[] projection,
                                  String sortOrder) throws FileNotFoundException {

    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 so on.
        includeFile(result, null, file);
    }
    return result;
}
```

实现 queryDocument

您实现的 `queryDocument()` 必须使用在 `DocumentsContract.Document` 中定义的列返回一个指向指定文件的 `Cursor`。

除了特定文件的信息外，`queryDocument()` 方法返回的信息与 `queryChildDocuments()` 中传递的信息相同：

```
@Override
public Cursor queryDocument(String documentId, String[] projection) throws
        FileNotFoundException {

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

实现 queryDocument

您必须实现 `openDocument()` 以返回表示指定文件的 `ParcelFileDescriptor`。其他应用可以使用返回的 `ParcelFileDescriptor` 来流式传输数据。用户选择了文件，并且客户端应用通过调用 `openFileDescriptor()` 来请求对文件的访问权限后，系统便会调用此方法。例如：

```
@Override
public ParcelFileDescriptor openDocument(final String documentId,
                                         final String mode,
                                         CancellationSignal signal) throws
        FileNotFoundException {
    Log.v(TAG, "openDocument, mode: " + mode);
    // It's OK to do network operations in this method to download the document,
    // as long as you periodically check the CancellationSignal. If you have an
    // extremely large file to transfer from the network, a better solution may
    // be pipes or sockets (see ParcelFileDescriptor for helper methods).

    final File file = getFileForDocId(documentId);

    final boolean isWrite = (mode.indexOf('w') != -1);
    if(isWrite) {
        // Attach a close listener if the document is opened in write mode.
        try {
            Handler handler = new Handler(getContext().getMainLooper());
            return ParcelFileDescriptor.open(file, accessMode, handler,
                        new ParcelFileDescriptor.OnCloseListener() {
                @Override
                public void onClose(IOException e) {

                    // Update the file with the cloud server. The client is done
                    // writing.
                    Log.i(TAG, "A file with id " +
                    documentId + " has been closed!
                    Time to " +
                    "update the server.");
                }

            });
        } catch (IOException e) {
            throw new FileNotFoundException("Failed to open document with id "
            + documentId + " and mode " + mode);
        }
    } else {
        return ParcelFileDescriptor.open(file, accessMode);
    }
}
```

安全性

假设您的文档提供程序是受密码保护的云存储服务，并且您想在开始共享用户的文件之前确保其已登录。如果用户未登录，您的应用应该执行哪些操作呢？ 解决方案是在您实现的 `queryRoots()` 中返回零个根目录。也就是空的根目录游标：

```
public Cursor queryRoots(String[] projection) throws FileNotFoundException {
...
    // If user is not logged in, return an empty root cursor.  This removes our
    // provider from the list entirely.
    if (!isUserLoggedIn()) {
        return result;
}
```

另一个步骤是调用 `getContentResolver().notifyChange()`。还记得 `DocumentsContract` 吗？ 我们将使用它来创建此 URI。 以下代码段会在每次用户的登录状态发生变化时指示系统查询文档提供程序的根目录。 如果用户未登录，则调用 `queryRoots()` 会返回一个空游标，如上文所示。 这可以确保只有在用户登录提供程序后其中的文档才可用。

```
private void onLoginButtonClick() {
    loginOrLogout();
    getContentResolver().notifyChange(DocumentsContract
            .buildRootsUri(AUTHORITY), null);
}
```