创建内容提供程序

本文内容

关键类

- `ContentProvider`
- `Cursor`
- `Uri`

相关示例

- 记事本示例应用

另请参阅

- 内容提供程序基础知识
- 日历提供程序

内容提供程序管理对中央数据存储区的访问。您将提供程序作为 Android 应用中的一个或多个类（连同清单文件中的元素）实现。 其中一个类会实现子类 `ContentProvider`，即您的提供程序与其他应用之间的接口。 尽管内容提供程序旨在向其他应用提供数据，但您的应用中必定有这样一些 Activity，它们允许用户查询和修改由提供程序管理的数据。

本主题的其余部分列出了开发内容提供程序的基本步骤和需要使用的 API。

着手开发前的准备工作

请在着手开发提供程序之前执行以下操作：

1. **决定您是否需要内容提供程序**。如果您想提供下列一项或多项功能，则需要开发内容提供程序：
   - 您想为其他应用提供复杂的数据或文件
   - 您想允许用户将复杂的数据从您的应用复制到其他应用中
   - 您想使用搜索框架提供自定义搜索建议

如果完全是在您自己的应用中使用，则*不*需要提供程序即可使用 SQLite 数据库。

2. 如果您尚未完成此项操作，请阅读内容提供程序基础知识主题，了解有关提供程序的详情。

接下来，请按照以下步骤开发您的提供程序：

1. 为您的数据设计原始存储。内容提供程序以两种方式提供数据：

   文件数据

   通常存储在文件中的数据，如照片、音频或视频。 将文件存储在您的应用的私有空间内。 您的提供程序可以应其他应用发出的文件请求提供文件句柄。

   “结构化”数据

   通常存储在数据库、数组或类似结构中的数据。 以兼容行列表的形式存储数据。行表示实体，如人员或库存项目。 列表示实体的某项数据，如人员的姓名或商品的价格。 此类数据通常存储在 SQLite 数据库中，但您可以使用任何类型的持久存储。 如需了解有关 Android 系统中提供的存储类型的更多信息，请参阅设计数据存储部分。

2. 定义 `ContentProvider` 类及其所需方法的具体实现。 此类是您的数据与 Android 系统其余部分之间的接口。 如需了解有关此类的详细信息，请参阅实现 ContentProvider 类部分。

3. 定义提供程序的授权字符串、其内容 URI 以及列名称。如果您想让提供程序的应用处理 Intent，则还要定义 Intent 操作、Extra 数据以及标志。 此外，还要定义想要访问您的数据的应用必须具备的权限。 您应该考虑在一个单独的协定类中将所有这些值定义为常量；以后您可以将此类公开给其他开发者。 如需了解有关内容 URI 的详细信息，请参阅设计内容 URI 部分。 如需了解有关 Intent 的详细信息，请参阅 Intent 和数据访问部分。

4. 添加其他可选部分，如示例数据或可以在提供程序与云数据之间同步数据的 `AbstractThreadedSyncAdapter` 实现。

设计数据存储

内容提供程序是用于访问以结构化格式保存的数据的接口。在您创建该接口之前，必须决定如何存储数据。 您可以按自己的喜好以任何形式存储数据，然后根据需要设计读写数据的接口。

以下是 Android 中提供的一些数据存储技术：

- Android 系统包括一个 SQLite 数据库 API，Android 自己的提供程序使用它来存储面向表的数据。 `SQLiteOpenHelper` 类可帮助您创建数据库，`SQLiteDatabase` 类是用于访问数据库的基类。

  请记住，您不必使用数据库来实现存储区。提供程序在外部表现为一组表，与关系型数据库类似，但这并不是对提供程序内部实现的要求；

- 为了存储文件数据，Android 提供了各种面向文件的 API。 如需了解有关文件存储的更多信息，请阅读数据存储主题。 如果您要设计提供媒体相关数据（如音乐或视频）的提供程序，则可开发一个合并了表数据和文件的提供程序。

- 要想使用基于网络的数据，请使用 `java.net` 和 `android.net` 中的类。 您也可以将基于网络的数据与本地数据存储（如数据库）同步，然后以表或文件的形式提供数据。 示例同步适配器示例应用展示了这类同步。

数据设计考虑事项

以下是一些设计提供程序数据结构的技巧：

- 表数据应始终具有一个“主键”列，提供程序将其作为与每行对应的唯一数字值加以维护。 您可以使用此值将该行链接到其他表中的相关行（将其用作“外键”）。 尽管您可以为此列使用任何名称 ，但使用 `BaseColumns._ID` 是最佳选择，因为将提供程序查询的结果链接到 `ListView` 的条件是，检索到的其中一个列的名称必须是 `_ID`；

- 如果您想提供位图图像或其他非常庞大的文件导向型数据，请将数据存储在一个文件中，然后间接提供这些数据，而不是直接将其存储在表中。 如果您执行了此操作，则需要告知提供程序的用户，他们需要使用 `ContentResolver` 文件方法来访问数据；

- 使用二进制大型对象 (BLOB) 数据类型存储大小或结构会发生变化的数据。 例如，您可以使用 BLOB 列来存储协议缓冲区或 JSON 结构。

  您也可以使用 BLOB 来实现*独立于架构*的表。在这类表中，您需要以 BLOB 形式定义一个主键列、一个 MIME 类型列以及一个或多个通用列。 这些 BLOB 列中数据的含义通过 MIME 类型列中的值指示。 这样一来，您就可以在同一个表中存储不同类型的行。 举例来说，联系人提供程序的“数据”表 `ContactsContract.Data` 便是一个独立于架构的表。

设计内容 URI

内容 URI 是用于在提供程序中标识数据的 URI。内容 URI 包括整个提供程序的符号名称（其**授权**）和一个指向表或文件的名称（**路径**）。 可选 ID 部分指向表中的单个行。 `ContentProvider` 的每一个数据访问方法都将内容 URI 作为参数；您可以利用这一点确定要访问的表、行或文件。

内容提供程序基础知识主题中描述了内容 URI 的基础知识。

设计授权

提供程序通常具有单一授权，该授权充当其 Android 内部名称。为避免与其他提供程序发生冲突，您应该使用互联网网域所有权（反向）作为提供程序授权的基础。 由于此建议也适用于 Android 软件包名称，因此您可以将提供程序授权定义为包含该提供程序的软件包名称的扩展名。例如，如果您的 Android 软件包名称为 `com.example.<appname>`，则应为提供程序提供 `com.example.<appname>.provider` 授权。

设计路径结构

开发者通常通过追加指向单个表的路径来根据权限创建内容 URI。 例如，如果您有两个表：*table1* 和 *table2*，则可以通过合并上一示例中的权限来生成 内容 URI `com.example.<appname>.provider/table1` 和 `com.example.<appname>.provider/table2`。路径并不限定于单个段，也无需为每一级路径都创建一个表。

处理内容 URI ID

按照惯例，提供程序通过接受末尾具有行所对应 ID 值的内容 URI 来提供对表中单个行的访问。 同样按照惯例，提供程序会将该 ID 值与表的 `_ID` 列进行匹配，并对匹配的行执行请求的访问。

这一惯例为访问提供程序的应用的常见设计模式提供了便利。应用会对提供程序执行查询，并使用 `CursorAdapter` 以 `ListView` 显示生成的 `Cursor`。 定义 `CursorAdapter` 的条件是， `Cursor` 中的其中一个列必须是 `_ID`

用户随后从 UI 上显示的行中选取其中一行，以查看或修改数据。 应用会从支持 `ListView` 的 `Cursor` 中获取对应行，获取该行的 `_ID` 值，将其追加到内容 URI，然后向提供程序发送访问请求。 然后，提供程序便可对用户选取的特定行执行查询或修改。

内容 URI 模式

为帮助您选择对传入的内容 URI 执行的操作，提供程序 API 加入了实用类 `UriMatcher`，它会将内容 URI“模式”映射到整型值。 您可以在一个 `switch` 语句中使用这些整型值，为匹配特定模式的一个或多个内容 URI 选择所需操作。

内容 URI 模式使用通配符匹配内容 URI：

- `*`：匹配由任意长度的任何有效字符组成的字符串
- `#`：匹配由任意长度的数字字符组成的字符串

以设计和编码内容 URI 处理为例，假设一个具有授权 `com.example.app.provider` 的提供程序能识别以下指向表的内容 URI：

- `content://com.example.app.provider/table1`：一个名为 `table1` 的表
- `content://com.example.app.provider/table2/dataset1`：一个名为 `dataset1` 的表
- `content://com.example.app.provider/table2/dataset2`：一个名为 `dataset2` 的表
- `content://com.example.app.provider/table3`：一个名为 `table3` 的表

提供程序也能识别追加了行 ID 的内容 URI，例如，`content://com.example.app.provider/table3/1` 对应由 `table3` 中 `1` 标识的行的内容 URI。

可以使用以下内容 URI 模式：

`content://com.example.app.provider/*`

匹配提供程序中的任何内容 URI。

`content://com.example.app.provider/table2/*`：

匹配表 `dataset1` 和表 `dataset2` 的内容 URI，但不匹配 `table1` 或 `table3` 的内容 URI。

`content://com.example.app.provider/table3/#`：匹配 `table3` 中单个行的内容 URI，如 `content://com.example.app.provider/table3/6` 对应由 `6` 标识的行的内容 URI。

以下代码段演示了 `UriMatcher` 中方法的工作方式。 此代码采用不同方式处理整个表的 URI 与单个行的 URI，它为表使用的内容 URI 模式是 `content://<authority>/<path>`，为单个行使用的内容 URI 模式则是 `content://<authority>/<path>/<id>`。

方法 `addURI()` 会将授权和路径映射到一个整型值。 方法 `match()` 会返回 URI 的整型值。`switch` 语句会在查询整个表与查询单个记录之间进行选择：

```
public class ExampleProvider extends ContentProvider {
...
    // Creates a UriMatcher object.
    private static final UriMatcher sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);

    static {
        /*
         * The calls to addURI() go here, for all of the content URI patterns that the provider
         * should recognize. For this snippet, only the calls for table 3 are shown.
         */

        /*
         * Sets the integer value for multiple rows in table 3 to 1. Notice that no wildcard is used
         * in the path
         */
        sUriMatcher.addURI("com.example.app.provider", "table3", 1);

        /*
         * Sets the code for a single row to 2. In this case, the "#" wildcard is
         * used. "content://com.example.app.provider/table3/3" matches, but
         * "content://com.example.app.provider/table3 doesn't.
         */
        sUriMatcher.addURI("com.example.app.provider", "table3/#", 2);
    }
...
    // Implements ContentProvider.query()
    public Cursor query(
        Uri uri,
        String[] projection,
        String selection,
        String[] selectionArgs,
        String sortOrder) {
...
        /*
         * Choose the table to query and a sort order based on the code returned for the incoming
         * URI. Here, too, only the statements for table 3 are shown.
         */
        switch (sUriMatcher.match(uri)) {


            // If the incoming URI was for all of table3
            case 1:

                if (TextUtils.isEmpty(sortOrder)) sortOrder = "_ID ASC";
                break;

            // If the incoming URI was for a single row
            case 2:

                /*
                 * Because this URI was for a single row, the _ID value part is
                 * present. Get the last path segment from the URI; this is the _ID value.
                 * Then, append the value to the WHERE clause for the query
                 */
                selection = selection + "_ID = " uri.getLastPathSegment();
                break;

            default:
            ...
                // If the URI is not recognized, you should do some error handling here.
        }
        // call the code to actually do the query
    }
```

另一个类 `ContentUris` 会提供一些工具方法，用于处理内容 URI 的 `id` 部分。`Uri` 类和 `Uri.Builder` 类包括一些工具方法，用于解析现有 `Uri` 对象和构建新对象。

实现 ContentProvider 类

`ContentProvider` 实例通过处理来自其他应用的请求来管理对结构化数据集的访问。 所有形式的访问最终都会调用 `ContentResolver`，后者接着调用 `ContentProvider` 的具体方法来获取访问权限。

必需方法

抽象类 `ContentProvider` 定义了六个抽象方法，您必须将这些方法作为自己具体子类的一部分加以实现。 所有这些方法（`onCreate()` 除外）都由一个尝试访问您的内容提供程序的客户端应用调用：

`query()`

从您的提供程序检索数据。使用参数选择要查询的表、要返回的行和列以及结果的排序顺序。 将数据作为 `Cursor` 对象返回。

`insert()`

在您的提供程序中插入一个新行。使用参数选择目标表并获取要使用的列值。 返回新插入行的内容 URI。

`update()`

更新您提供程序中的现有行。使用参数选择要更新的表和行，并获取更新后的列值。 返回已更新的行数。

`delete()`

从您的提供程序中删除行。使用参数选择要删除的表和行。 返回已删除的行数。

`getType()`

返回内容 URI 对应的 MIME 类型。实现内容提供程序 MIME 类型部分对此方法做了更详尽的描述。

`onCreate()`

初始化您的提供程序。Android 系统会在创建您的提供程序后立即调用此方法。 请注意，`ContentResolver` 对象尝试访问您的提供程序时，系统才会创建它。

请注意，这些方法的签名与同名的 `ContentResolver` 方法相同。

您在实现这些方法时应考虑以下事项：

- 所有这些方法（`onCreate()` 除外）都可由多个线程同时调用，因此它们必须是线程安全方法。如需了解有关多个线程的更多信息，请参阅进程和线程主题；
- 避免在 `onCreate()` 中执行长时间操作。将初始化任务推迟到实际需要时进行。 实现 onCreate() 方法部分对此做了更详尽的描述；
- 尽管您必须实现这些方法，但您的代码只需返回要求的数据类型，无需执行任何其他操作。 例如，您可能想防止其他应用向某些表插入数据。 要实现此目的，您可以忽略 `insert()` 调用并返回 0。

实现 query() 方法

`ContentProvider.query()` 方法必须返回 `Cursor` 对象。如果失败，则会引发 `Exception`。 如果您使用 SQLite 数据库作为数据存储，则只需返回由 `SQLiteDatabase` 类的其中一个 `query()` 方法返回的 `Cursor`。 如果查询不匹配任何行，您应该返回一个 `Cursor` 实例（其 `getCount()` 方法返回 0）。只有当查询过程中出现内部错误时，您才应该返回 `null`。

如果您不使用 SQLite 数据库作为数据存储，请使用 `Cursor` 的其中一个具体子类。 例如，在 `MatrixCursor` 类实现的游标中，每一行都是一个 `Object` 数组。 对于此类，请使用 `addRow()` 来添加新行。

请记住，Android 系统必须能够跨进程边界传播 `Exception`。 Android 可以为以下异常执行此操作，这些异常可能有助于处理查询错误：

- `IllegalArgumentException`（您可以选择在提供程序收到无效的内容 URI 时引发此异常）
- `NullPointerException`

实现 insert() 方法

`insert()` 方法会使用 `ContentValues` 参数中的值向相应表中添加新行。 如果 `ContentValues` 参数中未包含列名称，您可能想在您的提供程序代码或数据库架构中提供其默认值。

此方法应该返回新行的内容 URI。要想构建此方法，请使用 `withAppendedId()` 向表的内容 URI 追加新行的 `_ID`（或其他主键）值。

实现 delete() 方法

`delete()` 方法不需要从您的数据存储中实际删除行。 如果您将同步适配器与提供程序一起使用，应该考虑为已删除的行添加“删除”标志，而不是将行整个移除。 同步适配器可以检查是否存在已删除的行，并将它们从服务器中移除，然后再将它们从提供程序中删除。

实现 update() 方法

`update()` 方法采用 `insert()` 所使用的相同 `ContentValues` 参数，以及 `delete()` 和 `ContentProvider.query()` 所使用的相同 `selection` 和 `selectionArgs` 参数。 这样一来，您就可以在这些方法之间重复使用代码。

实现 onCreate() 方法

Android 系统会在启动提供程序时调用 `onCreate()`。您只应在此方法中执行运行快速的初始化任务，并将数据库创建和数据加载推迟到提供程序实际收到数据请求时进行。 如果您在 `onCreate()` 中执行长时间的任务，则会减慢提供程序的启动速度， 进而减慢提供程序对其他应用的响应速度。

例如，如果您使用 SQLite 数据库，可以在 `ContentProvider.onCreate()` 中创建一个新的 `SQLiteOpenHelper` 对象，然后在首次打开数据库时创建 SQL 表。 为简化这一过程，在您首次调用 `getWritableDatabase()` 时，它会自动调用 `SQLiteOpenHelper.onCreate()` 方法。

以下两个代码段展示了 `ContentProvider.onCreate()` 与 `SQLiteOpenHelper.onCreate()` 之间的交互。第一个代码段是 `ContentProvider.onCreate()` 的实现：

```
public class ExampleProvider extends ContentProvider

    /*
     * Defines a handle to the database helper object. The MainDatabaseHelper class is defined
     * in a following snippet.
     */
    private MainDatabaseHelper mOpenHelper;

    // Defines the database name
    private static final String DBNAME = "mydb";

    // Holds the database object
    private SQLiteDatabase db;

    public boolean onCreate() {

        /*
         * Creates a new helper object. This method always returns quickly.
         * Notice that the database itself isn't created or opened
         * until SQLiteOpenHelper.getWritableDatabase is called
         */
        mOpenHelper = new MainDatabaseHelper(
            getContext(),        // the application context
            DBNAME,              // the name of the database)
            null,                // uses the default SQLite cursor
            1                    // the version number
        );

        return true;
    }

    ...

    // Implements the provider's insert method
    public Cursor insert(Uri uri, ContentValues values) {
        // Insert code here to determine which table to open, handle error-checking, and so forth

        ...

        /*
         * Gets a writeable database. This will trigger its creation if it doesn't already exist.
         *
         */
        db = mOpenHelper.getWritableDatabase();
    }
}
```

下一个代码段是 `SQLiteOpenHelper.onCreate()` 的实现，其中包括一个帮助程序类：

```
...
// A string that defines the SQL statement for creating a table
private static final String SQL_CREATE_MAIN = "CREATE TABLE " +
    "main " +                       // Table's name
    "(" +                           // The columns in the table
    " _ID INTEGER PRIMARY KEY, " +
    " WORD TEXT"
    " FREQUENCY INTEGER " +
    " LOCALE TEXT )";
...
/**
 * Helper class that actually creates and manages the provider's underlying data repository.
 */
protected static final class MainDatabaseHelper extends SQLiteOpenHelper {

    /*
     * Instantiates an open helper for the provider's SQLite data repository
     * Do not do database creation and upgrade here.
     */
    MainDatabaseHelper(Context context) {
        super(context, DBNAME, null, 1);
    }

    /*
     * Creates the data repository. This is called when the provider attempts to open the
     * repository and SQLite reports that it doesn't exist.
     */
    public void onCreate(SQLiteDatabase db) {

        // Creates the main table
        db.execSQL(SQL_CREATE_MAIN);
    }
}
```

实现内容提供程序 MIME 类型

`ContentProvider` 类具有两个返回 MIME 类型的方法：

`getType()`

您必须为任何提供程序实现的必需方法之一。

`getStreamTypes()`

系统在您的提供程序提供文件时要求实现的方法。

表的 MIME 类型

`getType()` 方法会返回一个 MIME 格式的 `String`，后者描述内容 URI 参数返回的数据类型。`Uri` 参数可以是模式，而不是特定 URI；在这种情况下，您应该返回与匹配该模式的内容 URI 关联的数据类型。

对于文本、HTML 或 JPEG 等常见数据类型，`getType()` 应该为该数据返回标准 MIME 类型。 IANA MIME Media Types 网站上提供了这些标准类型的完整列表。

对于指向一个或多个表数据行的内容 URI，`getType()` 应该以 Android 供应商特有 MIME 格式返回 MIME 类型：

- 类型部分：`vnd`
- 子类型部分：
  - 如果 URI 模式用于单个行：`android.cursor.item/`
  - 如果 URI 模式用于多个行：`android.cursor.dir/`
- 提供程序特有部分：`vnd.<name>.<type>`

  您提供 `<name>` 和 `<type>`。 `<name>` 值应具有全局唯一性，`<type>` 值应在对应的 URI 模式中具有唯一性。 适合选择贵公司的名称或您的

应用 Android 软件包名称的某个部分作为 `<name>`。 适合选择 URI 关联表的标识字符串作为 `<type>`。

例如，如果提供程序的授权是 `com.example.app.provider`，并且它公开了一个名为 `table1` 的表，则 `table1` 中多个行的 MIME 类型是：

```
vnd.android.cursor.dir/vnd.com.example.provider.table1
```

对于 `table1` 的单个行，MIME 类型是：

```
vnd.android.cursor.item/vnd.com.example.provider.table1
```

文件的 MIME 类型

如果您的提供程序提供文件，请实现 `getStreamTypes()`。 该方法会为您的提供程序可以为给定内容 URI 返回的文件返回一个 MIME 类型 `String` 数组。 您应该通过 MIME 类型过滤器参数过滤您提供的 MIME 类型，以便只返回客户端想处理的那些 MIME 类型。

例如，假设提供程序以 `.jpg`、`.png` 和 `.gif` 格式文件形式提供照片图像。 如果应用调用 `ContentResolver.getStreamTypes()` 时使用了过滤器字符串 `image/*`（任何是“图像”的内容），则 `ContentProvider.getStreamTypes()` 方法应返回数组：

```
{ "image/jpeg", "image/png", "image/gif"}
```

如果应用只对 `.jpg` 文件感兴趣，则可以在调用 `ContentResolver.getStreamTypes()` 时使用过滤器字符串 `*\/jpeg`，`ContentProvider.getStreamTypes()` 应返回：

```
{"image/jpeg"}
```

如果您的提供程序未提供过滤器字符串中请求的任何 MIME 类型，则 `getStreamTypes()` 应返回 `null`。

实现协定类

协定类是一种 `public final` 类，其中包含对 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。 该类可确保即使 URI、列名称等数据的实际值发生变化，也可以正确访问提供程序，从而在提供程序与其他应用之间建立协定。

协定类对开发者也有帮助，因为其常量通常采用助记名称，因此可以降低开发者为列名称或 URI 使用错误值的可能性。 由于它是一种类，因此可以包含 Javadoc 文档。 集成开发环境（如 Android Studio）可以根据协定类自动完成常量名称，并为常量显示 Javadoc。

开发者无法从您的应用访问协定类的类文件，但他们可以通过您提供的 `.jar` 文件将其静态编译到其应用内。

举例来说，`ContactsContract` 类及其嵌套类便属于协定类。

实现内容提供程序权限

安全与权限主题中详细描述了 Android 系统各个方面的权限和访问。 数据存储主题也描述了各类存储实行中的安全与权限。 其中的要点简述如下：

- 默认情况下，存储在设备内部存储上的数据文件是您的应用和提供程序的私有数据文件；
- 您创建的 `SQLiteDatabase` 数据库是您的应用和提供程序的私有数据库；
- 默认情况下，您保存到外部存储的数据文件是*公用*并*可全局读取*的数据文件。 您无法使用内容提供程序来限制对外部存储内文件的访问，因为其他应用可以使用其他 API 调用来对它们执行读取和写入操作；
- 用于在您的设备的内部存储上打开或创建文件或 SQLite 数据库的方法调用可能会为所有其他应用同时授予读取和写入访问权限。 如果您将内部文件或数据库用作提供程序的存储区，并向其授予“可全局读取”或“可全局写入”访问权限，则您在清单文件中为提供程序设置的权限不会保护您的数据。 内部存储中文件和数据库的默认访问权限是“私有”，对于提供程序的存储区，您不应更改此权限。

如果您想使用内容提供程序权限来控制对数据的访问，则应将数据存储在内部文件、SQLite 数据库或“云”中（例如，远程服务器上），而且您应该保持文件和数据库为您的应用所私有。

实现权限

即使底层数据为私有数据，所有应用仍可从您的提供程序读取数据或向其写入数据，因为在默认情况下，您的提供程序未设置权限。 要想改变这种情况，请使用属性或 `<provider>` 元素的子元素在您的清单文件中为您的提供程序设置权限。 您可以设置适用于整个提供程序、特定表甚至特定记录的权限，或者设置同时适用于这三者的权限。

您可以通过清单文件中的一个或多个 `<permission>` 元素为您的提供程序定义权限。要使权限对您的提供程序具有唯一性，请为 `android:name` 属性使用 Java 风格作用域。 例如，将读取权限命名为 `com.example.app.provider.permission.READ_PROVIDER`。

以下列表描述了提供程序权限的作用域，从适用于整个提供程序的权限开始，然后逐渐细化。 更细化的权限优先于作用域较大的权限：

统一读写提供程序级别权限

> 一个同时控制对整个提供程序读取和写入访问的权限，通过 `<provider>` 元素的 `android:permission` 属性指定。

单独的读取和写入提供程序级别权限

> 针对整个提供程序的读取权限和写入权限。您可以通过 `<provider>` 元素的 `android:readPermission` 属性和 `android:writePermission` 属性 指定它们。它们优先于 `android:permission` 所需的权限。

路径级别权限

> 针对提供程序中内容 URI 的读取、写入或读取/写入权限。您可以通过 `<provider>` 元素的 `<path-permission>` 子元素指定您想控制的每个 URI。 对于您指定的每个内容 URI，您都可以指定读取/写入权限、读取权限或写入权限，或同时指定所有三种权限。 读取权限和写入权限优先于读取/写入权限。 此外，路径级别权限优先于提供程序级别权限。

临时权限

> 一种权限级别，即使应用不具备通常需要的权限，该级别也能授予对应用的临时访问权限。 临时访问功能可减少应用需要在其清单文件中请求的权限数量。 当您启用临时权限时，只有持续访问您的所有数据的应用才需要“永久性”提供程序访问权限。
>
> 假设您需要实现电子邮件提供程序和应用的权限，如果您想允许外部图像查看器应用显示您的提供程序中的照片附件， 为了在不请求权限的情况下为图像查看器提供必要的访问权限，可以为照片的内容 URI 设置临时权限。 对您的电子邮件应用进行相应设计，使应用能够在用户想要显示照片时向图像查看器发送一个 Intent，其中包含照片的内容 URI 以及权限标志。 图像查看器可随后查询您的电子邮件提供程序以检索照片，即使查看器不具备对您提供程序的正常读取权限，也不受影响。
>
> 要想启用临时权限，请设置 `<provider>` 元素的 `android:grantUriPermissions` 属性，或者向您的 `<provider>` 元素添加一个或多个 `<grant-uri-permission>` 子元素。如果您使用了临时权限，则每当您从提供程序中移除对某个内容 URI 的支持，并且该内容 URI 关联了临时权限时，都需要调用 `Context.revokeUriPermission()`。
>
> 该属性的值决定可访问的提供程序范围。 如果该属性设置为 `true`，则系统会向整个提供程序授予临时权限，该权限将替代您的提供程序级别或路径级别权限所需的任何其他权限。
>
> 如果此标志设置为 `false`，则您必须向 `<provider>` 元素添加 `<grant-uri-permission>` 子元素。每个子元素都指定授予的临时权限所对应的一个或多个内容 URI。
>
> 要向应用授予临时访问权限，Intent 必须包含 `FLAG_GRANT_READ_URI_PERMISSION` 和/或 `FLAG_GRANT_WRITE_URI_PERMISSION` 标志。 它们通过 `setFlags()` 方法进行设置。
>
> 如果 `android:grantUriPermissions` 属性不存在，则假设其为 `false`。

<Provider> 元素

与 `Activity` 和 `Service` 组件类似，必须使用 `<provider>` 元素在清单文件中为其应用定义 `ContentProvider` 的子类。 Android 系统会从该元素获取以下信息：

授权 (`android:authorities`)

> 用于在系统内标识整个提供程序的符号名称。设计内容 URI 部分对此属性做了更详尽的描述。

提供程序类名（ `android:name` ）

实现 `ContentProvider` 的类。实现 ContentProvider 类中对此类做了更详尽的描述。

权限

指定其他应用访问提供程序的数据所必须具备权限的属性：

- `android:grantUriPermssions`：临时权限标志
- `android:permission`：统一提供程序范围读取/写入权限
- `android:readPermission`：提供程序范围读取权限
- `android:writePermission`：提供程序范围写入权限

实现内容提供程序权限部分对权限及其对应属性做了更详尽的描述。

启动和控制属性

这些属性决定 Android 系统如何以及何时启动提供程序、提供程序的进程特性以及其他运行时设置：

- `android:enabled`：允许系统启动提供程序的标志。
- `android:exported`：允许其他应用使用此提供程序的标志。
- `android:initOrder`：此提供程序相对于同一进程中其他提供程序的启动顺序。
- `android:multiProcess`：允许系统在与调用客户端相同的进程中启动提供程序的标志。
- `android:process`：应在其中运行提供程序的进程的名称。
- `android:syncable`：指示提供程序的数据将与服务器上的数据同步的标志。

开发指南中针对 `<provider>` 元素的主题提供了这些属性的完整资料。

信息属性

提供程序的可选图标和标签：

- `android:icon`：包含提供程序图标的可绘制对象资源。 该图标出现在*Settings* > *Apps* > *All* 中应用列表内的提供程序标签旁；
- `android:label`：描述提供程序和/或其数据的信息标签。 该标签出现在*Settings* > *Apps* > *All*中的应用列表内。

开发指南中针对 `<provider>` 元素的主题提供了这些属性的完整资料。

Intent 和数据访问

应用可以通过 `Intent` 间接访问内容提供程序。 应用不会调用 `ContentResolver` 或 `ContentProvider` 的任何方法，它并不会直接提供，而是会发送一个启动某个 Activity 的 Intent，该 Activity 通常是提供程序自身应用的一部分。 目标 Activity 负责检索和显示其 UI 中的数据。 视 Intent 中的操作而定，目标 Activity 可能还会提示用户对提供程序的数据进行修改。 Intent 可能还包含目标 Activity 在 UI 中显示的“extra”数据；用户随后可以选择更改此数据，然后使用它来修改提供程序中的数据。

您可能想使用 Intent 访问权限来帮助确保数据完整性。您的提供程序可能依赖于根据严格定义的业务逻辑插入、更新和删除数据。 如果是这种情况，则允许其他应用直接修改您的数据可能会导致无效的数据。 如果您想让开发者使用 Intent 访问权限，请务必为其提供详尽的参考资料。向他们解释为什么使用自身应用 UI 的 Intent 访问比尝试通过代码修改数据更好。

处理想要修改您的提供程序数据的传入 Intent 与处理其他 Intent 没有区别。 您可以通过阅读 Intent 和 Intent 过滤器主题了解有关 Intent 用法的更多信息。