内容提供程序基础知识

本文内容

关键类

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

相关示例

- 游标（联系人）
- 游标（电话）

另请参阅

- 创建内容提供程序
- 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。提供程序是 Android 应用的一部分，通常提供自己的 UI 来使用数据。 但是，内容提供程序主要旨在供其他应用使用，这些应用使用提供程序客户端对象来访问提供程序。 提供程序与提供程序客户端共同提供一致的标准数据界面，该界面还可处理跨进程通信并保护数据访问的安全性。

本主题介绍了以下基础知识：

- 内容提供程序的工作方式。
- 用于从内容提供程序检索数据的 API。
- 用于在内容提供程序中插入、更新或删除数据的 API。

- 其他有助于使用提供程序的 API 功能。

概览

内容提供程序以一个或多个表（与在关系型数据库中找到的表类似）的形式将数据呈现给外部应用。 行表示提供程序收集的某种数据类型的实例，行中的每个列表示为实例收集的每条数据。

例如，Android 平台的内置提供程序之一是用户字典，它会存储用户想要保存的非标准字词的拼写。 表 1 描述了数据在此提供程序表中的显示情况：

表 1. 用户字典示例表格。

字词	应用 id	频率	语言区域	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5

在表 1 中，每行表示可能无法在标准词典中找到的字词实例。 每列表示该字词的某些数据，如该字词首次出现时的语言区域。 列标题是存储在提供程序中的列名称。 要引用行的语言区域，需要引用其 `locale` 列。对于此提供程序，`_ID` 列充当由提供程序自动维护的“主键”列。

> **注：**提供程序无需具有主键，也无需将 `_ID` 用作其主键的列名称（如果存在主键）。 但是，如果您要将来自提供程序的数据与 `ListView` 绑定，则其中一个列名称必须是 `_ID`。 显示查询结果部分详细说明了此要求。

访问提供程序

应用从具有 `ContentResolver` 客户端对象的内容提供程序访问数据。 此对象具有调用提供程序对象（`ContentProvider` 的某个具体子类的实例）中同名方法的方法。 `ContentResolver` 方法可提供持续存储的基本“CRUD”（创建、检索、更新和删除）功能。

客户端应用进程中的 `ContentResolver` 对象和拥有提供程序的应用中的 `ContentProvider` 对象可自动处理跨进程通信。 `ContentProvider` 还可充当其数据存储区和表格形式的数据外部显示之间的抽象层。

> **注：**要访问提供程序，您的应用通常需要在其清单文件中请求特定权限。 内容提供程序权限部分详细介绍了此内容。

例如，要从用户字典提供程序中获取字词及其语言区域的列表，则需调用 `ContentResolver.query()`。 `query()` 方法会调用用户字典提供程序所定义的 `ContentProvider.query()` 方法。 以下代码行显示了 `ContentResolver.query()` 调用：

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
    mProjection,                        // The columns to return for each row
    mSelectionClause                    // Selection criteria
    mSelectionArgs,                     // Selection criteria
    mSortOrder);                        // The sort order for the returned rows
```

表 2 显示了 `query(Uri,projection,selection,selectionArgs,sortOrder)` 的参数如何匹配 SQL SELECT 语句：

表 2. Query() 与 SQL 查询对比。

query() 参数	SELECT 关键字/参数	说明
`Uri`	`FROM table_name`	`Uri` 映射至提供程序中名为 *table_name* 的表。
`projection`	`col,col,col,...`	`projection` 是应该为检索到的每个行包含的列的数组。
`selection`	`WHERE col = value`	`selection` 会指定选择行的条件。
`selectionArgs`	（没有完全等效项。	

	选择参数会替换选择子句中 ? 占位符。）	
sortOrder	ORDER BY *col,col,...*	sortOrder 指定行在返回的 Cursor 中的显示顺序。

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**）和一个指向表的名称（**路径**）。 当您调用客户端方法来访问提供程序中的表时，该表的内容 URI 将是其参数之一。

在前面的代码行中，常量 `CONTENT_URI` 包含用户字典的“字词”表的内容 URI。 `ContentResolver` 对象会分析出 URI 的授权，并通过将该授权与已知提供程序的系统表进行比较，来“解析”提供程序。 然后， `ContentResolver` 可以将查询参数分派给正确的提供程序。

`ContentProvider` 使用内容 URI 的路径部分来选择要访问的表。 提供程序通常会为其公开的每个表显示一条**路径**。

在前面的代码行中，“字词”表的完整 URI 是：

```
content://user_dictionary/words
```

其中，`user_dictionary` 字符串是提供程序的授权，`words` 字符串是表的路径。 字符串 `content://`（**架构**）始终显示，并将此标识为内容 URI。

许多提供程序都允许您通过将 ID 值追加到 URI 末尾来访问表中的单个行。 例如，要从用户字典中检索 `_ID` 为 `4` 的行，则可使用此内容 URI：

```
Uri singleUri = ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI,4);
```

在检索到一组行后想要更新或删除其中某一行时通常会用到 ID 值。

> **注：**`Uri` 和 `Uri.Builder` 类包含根据字符串构建格式规范的 URI 对象的便利方法。 `ContentUris` 包含一些可以将 ID 值轻松追加到 URI 后的方法。 前面的代码段就是使用 `withAppendedId()` 将 ID 追加到 UserDictionary 内容 URI 后。

从提供程序检索数据

本节将以用户字典提供程序为例，介绍如何从提供程序中检索数据。

> 为了明确进行说明，本节中的代码段将在“UI 线程”上调用 `ContentResolver.query()`。但在实际代码中，您应该在单独线程上异步执行查询。 执行此操作的方式之一是使用 `CursorLoader` 类，加载器指南中对此有更为详细的介绍。 此外，前述代码行只是片段；它们不会显示整个应用。

要从提供程序中检索数据，请按照以下基本步骤执行操作：

1. 请求对提供程序的读取访问权限。
2. 定义将查询发送至提供程序的代码。

请求读取访问权限

要从提供程序检索数据，您的应需要具备对提供程序的“读取访问”权限。 您无法在运行时请求此权限；相反，您需要使用`<uses-permission>`元素和提供程序定义的准确权限名称，在清单文件中指明您需要此权限。 在您的清单文件中指定此元素后，您将有效地为应用“请求”此权限。 用户安装您的应用时，会隐式授予允许此请求。

要找出您正在使用的提供程序的读取访问权限的准确名称，以及提供程序使用的其他访问权限的名称，请查看提供程序的文档。

内容提供程序权限部分详细介绍了权限在访问提供程序过程中的作用。

用户字典提供程序在其清单文件中定义了权限 `android.permission.READ_USER_DICTIONARY`，因此希望从提供程序中进行读取的应用必需请求此权限。

构建查询

从提供程序中检索数据的下一步是构建查询。第一个代码段定义某些用于访问用户字典提供程序的变量：

```
// A "projection" defines the columns that will be returned for each row
String[] mProjection =
{
    UserDictionary.Words._ID,    // Contract class constant for the _ID column name
    UserDictionary.Words.WORD,   // Contract class constant for the word column name
    UserDictionary.Words.LOCALE  // Contract class constant for the locale column name
};

// Defines a string to contain the selection clause
String mSelectionClause = null;

// Initializes an array to contain selection arguments
String[] mSelectionArgs = {""};
```

下一个代码段以用户字典提供程序为例，显示了如何使用 `ContentResolver.query()`。 提供程序客户端查询与 SQL 查询类似，并且包含一组要返回的列、一组选择条件和排序顺序。

查询应该返回的列集被称为**投影**（变量 `mProjection`）。

用于指定要检索的行的表达式分割为选择子句和选择参数。 选择子句是逻辑和布尔表达式、列名称和值（变量 `mSelectionClause`）的组合。如果您指定了可替换参数 `?` 而非值，则查询方法会从选择参数数组（变量 `mSelectionArgs`）中检索值。

在下一个代码段中，如果用户未输入字词，则选择子句将设置为 `null`，而且查询会返回提供程序中的所有字词。 如果用户输入了字词，选择子句将设置为 `UserDictionary.Words.WORD + " = ?"` 且选择参数数组的第一个元素将设置为用户输入的字词。

```
/*
 * This defines a one-element String array to contain the selection argument.
 */
String[] mSelectionArgs = {""};

// Gets a word from the UI
mSearchString = mSearchWord.getText().toString();

// Remember to insert code here to check for invalid or malicious input.

// If the word is the empty string, gets everything
if (TextUtils.isEmpty(mSearchString)) {
    // Setting the selection clause to null will return all words
    mSelectionClause = null;
    mSelectionArgs[0] = "";

} else {
    // Constructs a selection clause that matches the word that the user entered.
    mSelectionClause = UserDictionary.Words.WORD + " = ?";

    // Moves the user's input string to the selection arguments.
    mSelectionArgs[0] = mSearchString;

}

// Does a query against the table and returns a Cursor object
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,  // The content URI of the words table
    mProjection,                       // The columns to return for each row
    mSelectionClause                   // Either null, or the word the user entered
    mSelectionArgs,                    // Either empty, or the string the user entered
    mSortOrder);                       // The sort order for the returned rows

// Some providers return null if an error occurs, others throw an exception
if (null == mCursor) {
    /*
     * Insert code here to handle the error. Be sure not to use the cursor! You may want to
     * call android.util.Log.e() to log this error.
     *
     */
// If the Cursor is empty, the provider found no matches
} else if (mCursor.getCount() < 1) {

    /*
     * Insert code here to notify the user that the search was unsuccessful. This isn't necessarily
     * an error. You may want to offer the user the option to insert a new row, or re-type the
     * search term.
     */

} else {
    // Insert code here to do something with the results

}
```

此查询与 SQL 语句相似：

```
SELECT _ID, word, locale FROM words WHERE word = <userinput> ORDER BY word ASC;
```

在此 SQL 语句中，会使用实际的列名称而非协定类常量。

防止恶意输入

如果内容提供程序管理的数据位于 SQL 数据库中，将不受信任的外部数据包括在原始 SQL 语句中可能会导致 SQL 注入。

考虑此选择子句：

```
// Constructs a selection clause by concatenating the user's input to the column name
String mSelectionClause =  "var = " + mUserInput;
```

如果您执行此操作，则会允许用户将恶意 SQL 串连到 SQL 语句上。 例如，用户可以为 `mUserInput` 输入“nothing; DROP TABLE *;”，这会生成选择子句 `var = nothing; DROP TABLE *;`。 由于选择子句是作为 SQL 语句处理，因此这可能会导致提供程序擦除基础 SQLite 数据库中的所有表（除非提供程序设置为可捕获 SQL 注入尝试）。

要避免此问题，可使用一个用于将 `?` 作为可替换参数的选择子句以及一个单独的选择参数数组。 执行此操作时，用户输入直接受查询约束，而不解释为 SQL 语句的一部分。 由于用户输入未作为 SQL 处理，因此无法注入恶意 SQL。请使用此选择子句，而不要使用串连来包括用户输入：

```
// Constructs a selection clause with a replaceable parameter
String mSelectionClause =  "var = ?";
```

按如下所示设置选择参数数组：

```
// Defines an array to contain the selection arguments
String[] selectionArgs = {""};
```

按如下所示将值置于选择参数数组中：

```
// Sets the selection argument to the user's input
selectionArgs[0] = mUserInput;
```

一个用于将 `?` 用作可替换参数的选择子句和一个选择参数数组是指定选择的首选方式，即使提供程序并未基于 SQL 数据库。

显示查询结果

`ContentResolver.query()` 客户端方法始终会返回符合以下条件的 `Cursor`：包含查询的投影为匹配查询选择条件的行指定的列。 `Cursor` 对象为其包含的行和列提供随机读取访问权限。 通过使用 `Cursor` 方法，您可以循环访问结果中的行、确定每个列的数据类型、从列中获取数据，并检查结果的其他属性。 某些 `Cursor` 实现会在提供程序的数据发生更改时自动更新对象和/或在 `Cursor` 更改时触发观察程序对象中的方法。

> **注：**提供程序可能会根据发出查询的对象的性质来限制对列的访问。 例如，联系人提供程序会限定只有同步适配器才能访问某些列，因此不会将它们返回至 Activity 或服务。

如果没有与选择条件匹配的行，则提供程序会返回 `Cursor.getCount()` 为 0（空游标）的 `Cursor` 对象。

如果出现内部错误，查询结果将取决于具体的提供程序。它可能会选择返回 `null`，或引发 `Exception`。

由于 `Cursor` 是行“列表”，因此显示 `Cursor` 内容的良好方式是通过 `SimpleCursorAdapter` 将其与 `ListView` 关联。

以下代码段将延续上一代码段的代码。它会创建一个包含由查询检索到的 `Cursor` 的 `SimpleCursorAdapter` 对象，并将此对象设置为 `ListView` 的适配器：

```
// Defines a list of columns to retrieve from the Cursor and load into an output row
String[] mWordListColumns =
{
    UserDictionary.Words.WORD,   // Contract class constant containing the word column name
    UserDictionary.Words.LOCALE  // Contract class constant containing the locale column name
};

// Defines a list of View IDs that will receive the Cursor columns for each row
int[] mWordListItems = { R.id.dictWord, R.id.locale};

// Creates a new SimpleCursorAdapter
mCursorAdapter = new SimpleCursorAdapter(
    getApplicationContext(),               // The application's Context object
    R.layout.wordlistrow,                  // A layout in XML for one row in the ListView
    mCursor,                               // The result from the query
    mWordListColumns,                      // A string array of column names in the cursor
    mWordListItems,                        // An integer array of view IDs in the row layout
    0);                                    // Flags (usually none are needed)

// Sets the adapter for the ListView
mWordList.setAdapter(mCursorAdapter);
```

注：要通过 `Cursor` 支持 `ListView`，游标必需包含名为 `_ID` 的列。 正因如此，前文显示的查询会为“字词”表检索 `_ID` 列，即使 `ListView` 未显示该列。 此限制也解释了为什么大多数提供程序的每个表都具有 `_ID` 列。

从查询结果中获取数据

您可以将查询结果用于其他任务，而不是仅显示它们。例如，您可以从用户字典中检索拼写，然后在其他提供程序中查找它们。 要执行此操作，您需要在 `Cursor` 中循环访问行：

```
// Determine the column index of the column named "word"
int index = mCursor.getColumnIndex(UserDictionary.Words.WORD);

/*
 * Only executes if the cursor is valid. The User Dictionary Provider returns null if
 * an internal error occurs. Other providers may throw an Exception instead of returning null.
 */

if (mCursor != null) {
    /*
     * Moves to the next row in the cursor. Before the first movement in the cursor, the
     * "row pointer" is -1, and if you try to retrieve data at that position you will get an
     * exception.
     */
    while (mCursor.moveToNext()) {

        // Gets the value from the column.
        newWord = mCursor.getString(index);

        // Insert code here to process the retrieved word.

        ...

        // end of while loop
    }
} else {

    // Insert code here to report an error if the cursor is null or the provider threw an exception.
}
```

`Cursor` 实现包含多个用于从对象中检索不同类型的数据的“获取”方法。 例如，上一个代码段使用 `getString()`。 它们还具有 `getType()` 方法，该方法会返回指示列的数据类型的值。

内容提供程序权限

提供程序的应用可以指定其他应用访问提供程序的数据所必需的权限。 这些权限可确保用户了解应用将尝试访问的数据。 根据提供程序的要求，其他应用会请求它们访问提供程序所需的权限。 最终用户会在安装应用时看到所请求的权限。

如果提供程序的应用未指定任何权限，则其他应用将无权访问提供程序的数据。 但是，无论指定权限为何，提供程序的应用中的组件始终具有完整的读取和写入访问权限。

如前所述，用户字典提供程序需要 `android.permission.READ_USER_DICTIONARY` 权限才能从中检索数据。 提供程序具有用于插入、更新或删除数据的单独 `android.permission.WRITE_USER_DICTIONARY` 权限。

要获取访问提供程序所需的权限，应用将通过其清单文件中的 `<uses-permission>` 元素来请求这些权限。Android 软件包管理器安装应用时，用户必须批准该应用请求的所有权限。 如果用户批准所有权限，软件包管理器将继续安装；如果用户未批准这些权限，软件包管理器将中止安装。

以下 `<uses-permission>` 元素会请求对用户字典提供程序的读取访问权限：

```
    <uses-permission android:name="android.permission.READ_USER_DICTIONARY">
```

安全与权限指南中详细介绍了权限对提供程序访问的影响。

插入、更新和删除数据

与从提供程序检索数据的方式相同，也可以通过提供程序客户端和提供程序 `ContentProvider` 之间的交互来修改数据。 您通过传递到 `ContentProvider` 的对应方法的参数来调用 `ContentResolver` 方法。 提供程序和提供程序客户端会自动处理安全性和跨进程通信。

插入数据

要将数据插入提供程序，可调用 `ContentResolver.insert()` 方法。此方法会在提供程序中插入新行并为该行返回内容 URI。 此代码段显示如何将新字词插入用户字典提供程序：

```
// Defines a new Uri object that receives the result of the insertion
Uri mNewUri;

...

// Defines an object to contain the new values to insert
ContentValues mNewValues = new ContentValues();

/*
 * Sets the values of each column and inserts the word. The arguments to the "put"
 * method are "column name" and "value"
 */
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "insert");
mNewValues.put(UserDictionary.Words.FREQUENCY, "100");

mNewUri = getContentResolver().insert(
    UserDictionary.Word.CONTENT_URI,   // the user dictionary content URI
    mNewValues                          // the values to insert
);
```

新行的数据会进入单个 `ContentValues` 对象中，该对象在形式上与单行游标类似。 此对象中的列不需要具有相同的数据类型，如果您不想指定值，则可以使用 `ContentValues.putNull()` 将列设置为 `null`。

代码段不会添加 `_ID` 列，因为系统会自动维护此列。 提供程序会向添加的每个行分配唯一的 `_ID` 值。 通常，提供程序会将此值用作表的主键。

`newUri` 中返回的内容 URI 会按照以下格式标识新添加的行：

```
content://user_dictionary/words/<id_value>
```

`<id_value>` 是新行的 `_ID` 内容。 大多数提供程序都能自动检测这种格式的内容 URI，然后在该特定行上执行请求的操作。

要从返回的 `Uri` 中获取 `_ID` 的值，请调用 `ContentUris.parseId()`。

更新数据

要更新行，请按照执行插入的方式使用具有更新值的 `ContentValues` 对象，并按照执行查询的方式使用选择条件。 您使用的客户端方法是 `ContentResolver.update()`。您只需将值添加至您要更新的列的 `ContentValues` 对象。 如果您要清除列的内容，请将值设置为 `null`。

以下代码段会将语言区域具有语言“en”的所有行的语言区域更改为 `null`。 返回值是已更新的行数：

```
// Defines an object to contain the updated values
ContentValues mUpdateValues = new ContentValues();

// Defines selection criteria for the rows you want to update
String mSelectionClause = UserDictionary.Words.LOCALE +  "LIKE ?";
String[] mSelectionArgs = {"en_%"};

// Defines a variable to contain the number of updated rows
int mRowsUpdated = 0;

...

/*
 * Sets the updated value and updates the selected words.
 */
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
    mUpdateValues                       // the columns to update
    mSelectionClause                    // the column to select on
    mSelectionArgs                      // the value to compare to
);
```

您还应该在调用 `ContentResolver.update()` 时检查用户输入。如需了解有关此内容的更多详情，请阅读防止恶意输入部分。

删除数据

删除行与检索行数据类似：为要删除的行指定选择条件，客户端方法会返回已删除的行数。 以下代码段会删除应用 ID 与“用户”匹配的行。该方法会返回已删除的行数。

```

// Defines selection criteria for the rows you want to delete
String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
String[] mSelectionArgs = {"user"};

// Defines a variable to contain the number of rows deleted
int mRowsDeleted = 0;

...

// Deletes the words that match the selection criteria
mRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
    mSelectionClause                    // the column to select on
    mSelectionArgs                      // the value to compare to
);
```

您还应该在调用 `ContentResolver.delete()` 时检查用户输入。如需了解有关此内容的更多详情，请阅读防止恶意输入部分。

提供程序数据类型

内容提供程序可以提供多种不同的数据类型。用户字典提供程序仅提供文本，但提供程序也能提供以下格式：

- 整型

- 长整型（长）
- 浮点型
- 长浮点型（双倍）

提供程序经常使用的另一种数据类型是作为 64KB 字节的数组实施的二进制大型对象 (BLOB)。 您可以通过查看 `Cursor` 类“获取”方法看到可用数据类型。

提供程序文档中通常都列出了其每个列的数据类型。 用户字典提供程序的数据类型列在其协定类 `UserDictionary.Words` 参考文档中（协定类部分对协定类进行了介绍）。 您也可以通过调用 `Cursor.getType()` 来确定数据类型。

提供程序还会维护其定义的每个内容 URI 的 MIME（多用途互联网邮件扩展）数据类型信息。您可以使用 MIME 类型信息查明应用是否可以处理提供程序提供的数据，或根据 MIME 类型选择处理类型。 在使用包含复杂数据结构或文件的提供程序时，通常需要 MIME 类型。 例如，联系人提供程序中的 `ContactsContract.Data` 表会使用 MIME 类型来标记每行中存储的联系人数据类型。 要获取与内容 URI 对应的 MIME 类型，请调用 `ContentResolver.getType()`。

MIME 类型引用部分介绍了标准和自定义 MIME 类型的语法。

提供程序访问的替代形式

提供程序访问的三种替代形式在应用开发过程中十分重要：

- 批量访问：您可以通过 `ContentProviderOperation` 类中的方法创建一批访问调用，然后通过 `ContentResolver.applyBatch()` 应用它们。
- 异步查询：您应该在单独线程中执行查询。执行此操作的方式之一是使用 `CursorLoader` 对象。 加载器指南中的示例展示了如何执行此操作。
- 通过 Intent 访问数据：尽管您无法直接向提供程序发送 Intent，但可以向提供程序的应用发送 Intent，后者通常具有修改提供程序数据的最佳配置。

下文将介绍通过 Intent 进行的批量访问和修改。

批量访问

批量访问提供程序适用于插入大量行，或通过同一方法调用在多个表中插入行，或者通常用于跨进程界限将一组操作作为事务处理（原子操作）执行。

要在“批量模式”下访问提供程序， 您可以创建 `ContentProviderOperation` 对象数组，然后使用 `ContentResolver.applyBatch()` 将其分派给内容提供程序。您需将内容提供程序的*授权*传递给此方法，而不是特定内容 URI。 这样可使数组中的每个 `ContentProviderOperation` 对象都能适用于其他表。 调用 `ContentResolver.applyBatch()` 会返回结果数组。

`ContactsContract.RawContacts` 协定类 的说明包括展示批量注入的代码段。 联系人管理器示例应用包含在其 `ContactAdder.java` 源文件中进行批量访问的示例。

通过 Intent 访问数据

Intent 可以提供对内容提供程序的间接访问。即使您的应用不具备访问权限，您也可以通过以下方式允许用户访问提供程序中的数据：从具有权限的应用中获取回结果 Intent，或者通过激活具有权限的应用，然后让用户在其中工作。

通过临时权限获取访问权限

即使您没有适当的访问权限，也可以通过以下方式访问内容提供程序中的数据：将 Intent 发送至具有权限的应用，然后接收回包含“URI”权限的结果 Intent。 这些是特定内容 URI 的权限，将持续至接收该权限的 Activity 结束。 具有永久权限的应用将通过在结果 Intent 中设置标志来授予临时权限：

- **读取权限：** `FLAG_GRANT_READ_URI_PERMISSION`
- **写入权限：** `FLAG_GRANT_WRITE_URI_PERMISSION`

> **使用帮助程序应用显示数据**
>
> 如果您的应用*具有*访问权限，您可能仍想使用 Intent 对象在其他应用中显示数据。 例如，日历应用接受 `ACTION_VIEW` Intent 对象，用于显示特定的日期或事件。 这样，您可以在不创建自己的 UI 的情况下显示日历信息。 如需了解有关此功能的更多信息，请参见日历提供程序指南。
>
> 您向其发送 Intent 对象的应用不必是与提供程序关联的应用。 例如，您可以从联系人提供程序中检索联系人，然后将包含联系人图像的内容 URI 的 `ACTION_VIEW` Intent 发送至图像查看器。

> **注：**这些标志不会为其授权包含在内容 URI 中的提供程序提供常规的读取或写入访问权限。 访问权限仅适用于 URI 本身。

提供程序使用 `<provider>` 元素的 `android:grantUriPermission` 属性以及 `<provider>` 元素的 `<grant-uri-permission>` 子元素在其清单文件中定义内容 URI 的 URI 权限。**安全与权限**指南中“URI 权限”部分更加详细地说明了 URI 权限机制。

例如，即使您没有 `READ_CONTACTS` 权限，也可以在联系人提供程序中检索联系人的数据。您可能希望在向联系人发送电子生日祝福的应用中执行此操作。 您更愿意让用户控制应用所使用的联系人，而不是请求 `READ_CONTACTS`，让您能够访问用户的所有联系人及其信息。 要执行此操作，您需要使用以下进程：

1. 您的应用会使用方法 `startActivityForResult()` 发送包含操作 `ACTION_PICK` 和“联系人”MIME 类型 `CONTENT_ITEM_TYPE` 的 Intent 对象。
2. 由于此 Intent 与“联系人”应用的“选择” Activity 的 Intent 过滤器相匹配，因此 Activity 会显示在前台。
3. 在选择 Activity 中，用户选择要更新的联系人。 发生此情况时，选择 Activity 会调用 `setResult(resultcode, intent)` 以设置用于返回至应用的 Intent。Intent 包含用户选择的联系人的内容 URI，以及“extra”标志 `FLAG_GRANT_READ_URI_PERMISSION`。这些标志会为您的应用授予读取内容 URI 所指向联系人数据的 URI 权限。 然后，选择 Activity 会调用 `finish()`， 将控制权交还给您的应用。
4. 您的 Activity 会返回至前台，系统会调用您的 Activity 的 `onActivityResult()` 方法。此方法会收到“联系人”应用中选择 Activity 所创建的结果 Intent。
5. 通过来自结果 Intent 的内容 URI，您可以读取来自联系人提供程序的联系人数据，即使您未在清单文件中请求对该提供程序的永久读取访问权限。 您可以获取联系人的生日信息或其电子邮件地址，然后发送电子祝福。

使用其他应用

允许用户修改您无权访问的数据的简单方法是激活具有权限的应用，让用户在其中执行工作。

例如，日历应用接受 `ACTION_INSERT` Intent 对象，这让您可以激活 应用的插入 UI。您可以在此 Intent（应用将使用该 Intent 来预先填充 UI）中传递“extra”数据。 由于定期事件具有复杂的语法，因此将事件插入日历提供程序的首选方式是激活具有 `ACTION_INSERT` 的日历应用，然后让用户在其中插入事件。

协定类

协定类定义帮助应用使用内容 URI、列名称、 Intent 操作以及内容提供程序的其他功能的常量。 协定类未自动包含在提供程序中；提供程序的开发者需要定义它们，然后使其可用于其他开发者。 Android 平台中包含的许多提供程序都在软件包 `android.provider` 中具有对应的协定类。

例如，用户字典提供程序具有包含内容 URI 和列名称常量的协定类 `UserDictionary`。 “字词”表的内容 URI 在常量 `UserDictionary.Words.CONTENT_URI` 中定义。 `UserDictionary.Words` 类也包含列名称常量，本指南的示例代码段中就使用了该常量。例如，查询投影可以定义为：

```
String[] mProjection =
{
    UserDictionary.Words._ID,
    UserDictionary.Words.WORD,
    UserDictionary.Words.LOCALE
};
```

联系人提供程序的 `ContactsContract` 也是一个协定类。 此类的参考文档包括示例代码段。其子类之一 `ContactsContract.Intents.Insert` 是包含 Intent 和 Intent 数据的协定类。

MIME 类型引用

内容提供程序可以返回标准 MIME 媒体类型和/或自定义 MIME 类型字符串。

MIME 类型具有格式

```
type/subtype
```

例如，众所周知的 MIME 类型 `text/html` 具有 `text` 类型和 `html` 子类型。如果提供程序为 URI 返回此类型，则意味着使用该 URI 的查询会

返回包含 HTML 标记的文本。

自定义 MIME 类型字符串（也称为“特定于供应商”的 MIME 类型）具有更加复杂的*类型*和*子类型*值。 *类型*值始终为

```
vnd.android.cursor.dir
```

（多行）或

```
vnd.android.cursor.item
```

（单行）。

*子类型*特定于提供程序。Android 内置提供程序通常具有简单的子类型。 例如，当“通讯录”应用为电话号码创建行时，它会在行中设置以下 MIME 类型：

```
vnd.android.cursor.item/phone_v2
```

请注意，子类型值只是 `phone_v2`。

其他提供程序开发者可能会根据提供程序的授权和表名称创建自己的子类型模式。 例如，假设提供程序包含列车时刻表。 提供程序的授权是 `com.example.trains`，并包含表 Line1、Line2 和 Line3。在响应表 Line1 的内容 URI

```
content://com.example.trains/Line1
```

时，提供程序会返回 MIME 类型

```
vnd.android.cursor.dir/vnd.example.line1
```

在响应表 Line2 中第 5 行的内容 URI

```
content://com.example.trains/Line2/5
```

时，提供程序会返回 MIME 类型

```
vnd.android.cursor.item/vnd.example.line2
```

大多数内容提供程序都会为其使用的 MIME 类型定义协定类常量。例如，联系人提供程序协定类 `ContactsContract.RawContacts` 会为单个原始联系人行的 MIME 类型定义常量 `CONTENT_ITEM_TYPE`。

内容 URI 部分介绍了单个行的内容 URI。