

## Android 部分 ContentProvider 篇

### 1. 内容提供者是什么？

内容提供者 (Content Provider) 主要用于在不同的应用程序之间实现数据共享的功能，它提供了一套完整的机制，允许一个程序访问另一个程序中的数据，同时还能保证被访数据的安全性。目前，使用内容提供者是Android 实现跨程序共享数据的标准方式。

不同于文件存储和 SharedPreferences 存储中的两种全局可读可写操作模式，内容提供者可以选择只对哪一部分数据进行共享，从而保证我们程序中的隐私数据不会泄露的风险。

### 2. 内容提供者的使用

我们一般用内容提供者都是用来查询数据的：

```
1 Cursor cursor = getContentResolver().query(final Uri uri, String[] projection, String
    selection, String[] selectionArgs, String sortOrder, CancellationSignal cancellationSignal)
```

- uri，指定查询某一个程序下的某一张表
- projection，指定查询的列名
- selection，指定查询条件，相当于 sql 语句中 where 后面的条件
- selectionArgs，给selection 中的占位符提供具体的值
- orderBy，指定查询结果排序方式
- cancellationSignal，取消正在进行操作信号量

写过SQLite 代码的你一定对此方法非常熟悉吧！等你看完后面 ContentProvider 原理机制的时候，一定会恍然大悟吧！

想要访问内容提供者中共享的数据，就一定要借助 ContentResolver 类，可以通过 Context 中的getContentResolver()方法获取该类的实例。ContentResolver 中提供了一系列的方法用于对数据进行 CRUD（增删改查）操作，其中insert()方法用于添加数据，update()方法用于数据的更新，delete()方法用于数据的删除，query()方法用于数据的查询。这好像SQLite 数据库操作有木有？

不同于 SQLiteDatabase, ContentResolver 中的增删改查方法都是不接收表名参数的，而是使用一个 Uri 的参数代替，这个参数被称作内容 URI。内容 URI 给内容提供者中的数据建立了唯一的标识符，它主要由两部分组成：authority 和 path。authority 是用于对不同的应用程序做区分的，一般为了避免冲突，都会采用程序包名的方式来进行命名。比如某个程序的包名为 com.example.app, 那么该程序对应的 authority 就可以命名为

`com.example.app.provider`。`path` 则是用于对同一应用程序中不同的表做区分的，通常都会添加到 `authority` 的后面。比如某个程序的数据库里存在两张表：`table1` 和 `table2`，这时就可以将 `path` 分别命名为 `/table1` 和 `/table2`，然后把 `authority` 和 `path` 进行组合，内容的 `URI` 就变成了 `com.example.app.provider/table1` 和 `com.example.app.provider/table2`。不过目前还是很难辨认出这两个字符串就是两个内容 `URI`，我们还需要在字符串的头部加上协议声明。因此，内容 `URI` 最标准的格式写法如下：

```
2. content://com.example.app.provider/table2
1. content://com.example.app.provider/table1
```

在得到内容 `URI` 字符串之后，我们还需要将它解析成 `Uri` 对象才可以作为参数传入。解析的方法也相当简单，代码如下所示：

```
1. Uri uri = new Uri.parse("content://com.example.app.provider/table1");
```

只需要调用 `Uri` 的静态方法 `parse()` 就可以把内容 `URI` 字符串解析成 `URI` 对象。现在，我们可以通过这个 `Uri` 对象来查询 `table1` 表中的数据了。代码如下所示：

```
1. Cursor cursor = getContentResolver()
2.         .query(
3.             uri,projection,selection,selectionArgs,sortOrder
4.         );
```

`query()` 方法接收的参数跟 `SQLiteDatabase` 中的 `query()` 方法接收的参数很像，但总体来说这个稍微简单一些，毕竟这是在访问其他程序中的数据，没必要构建复杂的查询语句。下标对内容提供者中的 `query` 的接收的参数进行了详细的解释：

查询完成仍然会返回一个 `Cursor` 对象，这时我们就可以将数据从 `Cursor` 对象中逐个读取出来了。读取的思路仍然是对这个 `Cursor` 对象进行遍历，然后一条一条的取出数据即可，代码如下：

```
1. if(cursor != null){//注意这里一定要进行一次判空，因为有可能你要查询的表根本不存在
2.     while(cursor.moveToNext()){
3.         String column1 = cursor.getString(cursor.getColumnIndex("column1"));
4.         int column2 = cursor.getInt(cursor.getColumnIndex("column2"));
5.     }
6. }
```

查询都会了，那么剩下的增加，删除，修改自然也不在话下了，代码如下所示：

1. //增加数据

淘宝搜《闵课通商学院》、小白轻松拿高薪offer

```

2. ContentValues values = new ContentValues();
3. values.put("Column1","text");
4. values.put("Column2","1");
5. getContextMenu.insert(uri,values);
6.
7. //删除数据
8. getContextMenu.delete(uri,"column2 = ?",new String[]{ "1" });
9.
10. //更新数据
11. ContentValues values = new ContentValues();
12. values.put("Column1","改数据");
13. getContextMenu.update(uri,values,"column1 = ?
    and column2 = ?",new String[]{"text","1"});
    
```

### 3. 如何创建属于自己应用的内容提供者？

前面已经提到过，如果要想实现跨程序共享数据的功能，官方推荐的方式就是使用内容提供者，可以新建一个类去继承 `ContentProvider` 类的方式来创建一个自己的内容提供者。`ContentProvider` 类有6个抽象方法，我们在使用子类继承它的时候，需要将这6个方法全部重写。新建 `MyProvider` 继承 `ContentProvider` 类，代码如下所示：

```

1. public class MyProvider extends ContentProvider {
2.
3.     @Override
4.     public boolean onCreate() {
5.         return false;
6.     }
7.
8.     @Override
9.     public Cursor query(Uri uri, String[] projection, String selection,
10.                        String[] selectionArgs, String sortOrder) {
11.         return null;
12.     } //查询
13.
14.     @Override
15.     public Uri insert(Uri uri, ContentValues values) {
16.         return null;
    
```

```
17.     }//添加
18.
19.     @Override
20.     public int update(Uri uri, ContentValues values, String selection,
21.                        String[] selectionArgs) {
22.         return 0;
```

淘宝搜《[闵课通商学院](#)》，小白轻松拿高薪offer

```

23.     }//更新
24.
25.     @Override
26.     public int delete(Uri uri, String selection, String[] selectionArgs) {
27.         return 0;
28.     }//删除
29.
30.     @Override
31.     public String getType(Uri uri) {
32.         return null;
33.     }
34. }
    
```

在这6个方法中，相信出来增删改查的方法你知道之外，剩下两个方法你可能不知道，下面就对这些方法进行一一介绍：

#### 1.onCreate()方法：

初始化内容提供器的时候调用。通常会在这里完成对数据库的创建和升级等操作。返回 **true** 表示内容提供器初始化成功，返回 **false** 则表示失败。注意，只有当存在 **ContentResolver** 尝试访问我们的程序中的数据时，内容提供器才会被初始化。

#### 2.query()方法：

从内容提供器中查询数据。使用 **uri** 参数来确定查询的哪张表，**projection** 参数用于确定查询的哪一列，**selection** 和 **selectionArgs** 参数用于约束查询哪些行，**sortOrder** 参数用于对结果进行排序，查询的结果存放在 **Cursor** 对象中返回。

#### 3.insert()方法：

向内容提供器中添加一条数据。使用 **uri** 参数来确定要添加的表，待添加的数据保存在 **values** 参数中。添加完成后，返回一个用于表示这条新纪录的 **URI**。

#### 4.update()方法：

更新内容提供器中已有的数据。使用 **uri** 参数来确定更新哪一张表中的数据，新数据保存着 **values** 参数当中，**selection** 和 **selectionArgs** 参数用于约束更新哪些行，受影响的行数将作为返回值返回。

#### 5.delete()方法：

从内容提供器中删除数据。使用 **uri** 参数来确定删除哪一张表中的数据，**selection** 和 **selectionArgs** 参数用于约束删除哪些行，被删除的行数将作为返回值返回。

#### 6.getType()方法：

根据传入的内容 **URI** 来返回相应的 **MIME** 类型。

可以看到，几乎每一个方法都会带有 `Uri` 这个参数，这个参数也正是调用 `ContentResolver` 的增删改查方法时传递过来的。而现在，我们需要对传入的 `Uri` 参数进行

淘宝搜《[闵课通商学院](#)》，小白轻松拿高薪offer

解析，从中分析出调用方期望访问的表和数据。

回顾一下，一个标准的内容 URI 写法是这样的：

```
1 content://com.example.app.provider/table1
```

这就表示调用方期望访问的是 `com.example.app` 这个应用的 `table1` 表中的数据。除此之外，我们还可以在这个内容 URI 的后面加上一个 `id`，如下所示：

```
1. content://com.example.app.provider/table1/1
```

这就表示调用方期望访问的是 `com.example.app` 这个应用的 `table1` 表中 `id` 为 1 的数据。

内容 URI 的格式主要有以上两种，以路径结尾就表示期望访问该表中所有的数据，以 `id` 结果就表示期望访问该表中拥有相应 `id` 的数据。我们可以使用通配符的方式来分别匹配这两中格式的内容 URI，规则如下：

\*：表示匹配任意长度的任意字符。

#：表示匹配任意长度的任意数字。

所以，一个能够匹配任意表的内容 URI 格式就可以写成：

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

而一个能够匹配 `table` 表中任意一行数据的内容 URI 格式就可以写成：

```
1. content://com.example.app.provider/table1/#
```

接着，我们再借助 `UriMatcher` 这个类就可以轻松地实现匹配内容 URI 的功能。`UriMatcher` 中提供了一个 `addURI()` 方法，这个方法接收三个参数，可以分别把 `authority`、`path` 和一个自定义代码传进去，这个自定义代码其实就是一个 `final` 的 `int` 类型的具值。这样，当调用 `UriMatcher` 的 `match()` 方法时，就可以将一个 `Uri` 对象传入，返回值是某个能够匹配这个 `Uri` 对象所对应的自定义代码，利用这个代码，我们就可以判断出调用方期望访问的是哪张表中的数据了。修改上述的 `MyProvider` 代码如下所示：

```
1. public class MyProvider extends ContentProvider {
2.
3.     public static final int TABLE1_DIR = 0;
4.
5.     public static final int TABLE1_ITEM = 1;
6. }
```



```
7.      public static final int TABLE2_DIR = 2;
8.
9.      public static final int TABLE2_ITEM = 3;
10.
11.     private static UriMatcher uriMatcher;
12.
```

淘宝搜《[闵课通商学院](#)》，小白轻松拿高薪offer

```

13.
    static {
14.
15.         uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
16.         uriMatcher.addURI("com.example.app.provider", "table1", TABLE1_DIR);
17.         uriMatcher.addURI("com.example.app.provider", "table1/#", TABLE1_ITEM)
18.         ;
19.         uriMatcher.addURI("com.example.app.provider", "table2", TABLE2_DIR);
20.         uriMatcher.addURI("com.example.app.provider", "table2/#", TABLE2_ITEM)
21.         ;
22.     }
23.
24.     ...
25.
26.     @Override
27.     public Cursor query(Uri uri, String[] projection, String selection,
28.                         String[] selectionArgs, String sortOrder) {
29.         switch(uriMatcher.match(uri)){
30.             case TABLE1_DIR:
31.                 //查询table1中的数据
32.                 break;
33.
34.             case TABLE1_ITEM:
35.                 //查询table1中的单条数据
36.                 break;
37.
38.             case TABLE2_DIR:
39.                 //查询table2中的数据
40.                 break;
41.

```

```
42.         case TABLE2_ITEM:
43.             //查询table2中的单条数据
44.             break;
45.
46.     }
47.
48.
49.     return null; 50.
51. }//查询
52.
53. ...
54. }
```

淘宝搜《[闵课通商学院](#)》、小白轻松拿高薪offer

上述代码只是以 `query()` 方法为例做了个示范，其实 `insert()` 方法，`update()`、`delete()` 方法的实现也是跟 `query()` 方法是差不多的，它们都会携带 `Uri` 这个参数，然后同样利用 `UriMatcher` 的 `match()` 方法判断出调用方期望访问的是哪张表，再对该表中的数据进行相应的操作就可以了。

除此之外，还有个方法你比较陌生，这个方法就是 `getType()` 方法。它是所有的内容提供者都必须提供的一个方法，用于获取 `Uri` 对象所对应的 MIME 类型。一个内容 `URI` 所对应的 MIME 字符串主要由 3 部分组成，Android 对这 3 个部分做了如下格式规定：

- 必须以 `vnd` 开头
- 如果内容 `URI` 以路径结尾，则后接 `android.cursor.dir/`，如果内容 `URI` 以 `id` 结尾，则后接 `android.cursor.item/`。
- 最后接 `vnd.< authority >.< path >`

所以，对于 `content://com.example.app.provider/table1` 这个内容 `URI`，它所对应的 MIME 类型就可以写成：

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

对于 `content://com.example.app.provider/table1/1` 这个内容 `URI`，它所对应的 MIME 类型就可以写成：

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

现在，我们可以继续完善 `MyProvider` 类中的内容，这次实现 `getType()` 方法的逻辑，代码如下：

```
1. public class MyProvider extends ContentProvider{
2.
3.     ...
4.
5.     @Override
6.     public String getType(Uri uri){
7.         switch(uriMatcher.match(uri)){
8.             case TABLE1_DIR:
9.                 return "vnd.android.cursor.dir/vnd.com.example.app.provider.
table1"
10.                break;
11.
12.             case TABLE1_ITEM:
13.                 return "vnd.android.cursor.item/vnd.com.example.app.provider
.table1 "
14.                break;
15.
16.             case TABLE2_DIR:
17.                 return "vnd.android.cursor.dir/vnd.com.example.app.provider
.table2"
```

```

18.             break;
19.
20.         case TABLE2_ITEM:
21.             return "vnd.android.cursor.item/vnd.com.example.app.provider
                .table2 "
22.             break;
23.         }
24.     }
25.
26.     ...
27.
28. }
```

到这里，一个完整的内容提供者就创建完成了，现在任何一个应用程序都可以使用 **ContentResolver** 来访问我们程序中的数据。那么前面所提到的，如何才能保证隐私数据不会泄漏出去呢？其实多亏了内容提供器的良好机制，这个问题在不知不觉中已经被解决了。因为所有的 **CRUD** 操作都一定要匹配到相应内容 **URI** 格式才能进行的，而我们当然不可能向 **UriMatcher** 中添加隐私数据的 **URI**，所以这部分数据根本无法被外部程序访问到，安全问题也就不存在了。

好了，创建内容提供器的步骤你也已经清楚了，下面就来实战一下，真正体验一回跨程序数据共享的功能。

还没有结束呢？我们都知道 4 大组件都需要在 **AndroidManifest.xml** 文件中进行注册，既然完成的内容提供者写好了，那么下一步就是去 **AndroidManifest.xml** 文件中进行注册，然后这个内容提供者就可以使用了。我们就来拿一个例子进行讲解，一个标准的内容提供器的封装代码如下所示：

```

1. public class DatabaseProvider extends ContentProvider {
2.
3.     public static final int BOOK_DIR = 0;
4.
5.     public static final int BOOK_ITEM = 1;
6.
7.     public static final int CATEGORY_DIR = 2;
8.
9.     public static final int CATEGORY_ITEM = 3;
10.
11.     public static final String AUTHORITY = "com.example.databasetest.provide
    r";
12.
13.     private static UriMatcher uriMatcher;
14.
15.     private MyDatabaseHelper dbHelper; //内容提供者数据库支持
```

```

16.
17.     static {
18.         uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
19.         uriMatcher.addURI(AUTHORITY, "book", BOOK_DIR);
20.         uriMatcher.addURI(AUTHORITY, "book/#", BOOK_ITEM);
21.         uriMatcher.addURI(AUTHORITY, "category", CATEGORY_DIR);
22.         uriMatcher.addURI(AUTHORITY, "category/#", CATEGORY_ITEM);
23.     }
24.
25.     @Override
26.     public boolean onCreate() {
27.         dbHelper = new MyDatabaseHelper(getContext(), "BookStore.db", null,
28.             2); // 创建内容提供者要使用的数据库
29.         return true; // 这里一定返回 true，不然内容提供者无法被使用
30.     }
31.
32.     @Override
33.     public Cursor query(Uri uri, String[] projection, String selection,
34.         String[] selectionArgs, String sortOrder) {
35.         // 查询数据
36.         SQLiteDatabase db = dbHelper.getReadableDatabase();
37.         Cursor cursor = null;
38.         switch (uriMatcher.match(uri)) {
39.             case BOOK_DIR:
40.                 cursor = db.query("Book", projection, selection, selectionArgs,
41.                     null, null, sortOrder);
42.                 break;
43.             case BOOK_ITEM:
44.                 String bookId = uri.getPathSegments().get(1);
45.                 cursor = db.query("Book", projection, "id = ?", new String[]
46.                     { bookId }, null, null, sortOrder);
47.                 break;
48.
49.             case CATEGORY_DIR:
50.                 cursor = db.query("Category", projection, selection,

```

```
51.                selectionArgs, null, null, sortOrder);
52.                break;
53.
54.            case CATEGORY_ITEM:
55.                String categoryId = uri.getPathSegments().get(1);
56.                cursor = db.query("Category", projection, "id = ?", new String[
                    ]
```

淘宝搜《闵课通商学院》，小白轻松拿高薪offer

```

57.         { categoryId }, null, null, sortOrder);
58.         break;
59.
60.     default:
61.         break;
62.     }
63.
64.     return cursor;
65. }
66.
67. @Override
68. public Uri insert(Uri uri, ContentValues values) {
69.     // 添加数据
70.     SQLiteDatabase db = dbHelper.getWritableDatabase();
71.     Uri uriReturn = null;
72.     switch (uriMatcher.match(uri)) {
73.         case BOOK_DIR:
74.
75.         case BOOK_ITEM:
76.             long newBookId = db.insert("Book", null, values);
77.             uriReturn = Uri.parse("content://" + AUTHORITY + "/book/" +
78.                                     newBookId);
79.             break;
80.
81.         case CATEGORY_DIR:
82.
83.         case CATEGORY_ITEM:
84.             long newCategoryId = db.insert("Category", null, values);
85.             uriReturn = Uri.parse("content://" + AUTHORITY + "/category/"
86.                                     +
87.                                     newCategoryId);
88.             break;
89.         default:

```



```
90.         break;
91.     }
92.
93.     return uriReturn;
94. }
95.
96. @Override
97. public int update(Uri uri, ContentValues values, String selection,
98.                   String[] selectionArgs) {
```

淘宝搜《闵课通商学院》，小白轻松拿高薪offer

```

99.         // 更新数据
100.         SQLiteDatabase db = dbHelper.getWritableDatabase();
101.         int updatedRows = 0;
102.         switch (uriMatcher.match(uri)) {
103.
104.             case BOOK_DIR:
105.                 updatedRows = db.update("Book", values, selection, selectionArgs);
106.                 break;
107.
108.             case BOOK_ITEM:
109.                 String bookId = uri.getPathSegments().get(1);
110.                 updatedRows = db.update("Book", values, "id = ?", new String[]
111.
112.                                     { bookId });
113.
114.                 break;
115.
116.             case CATEGORY_DIR:
117.                 updatedRows = db.update("Category", values, selection,
118.
119.                                     selectionArgs);
120.
121.                 break;
122.
123.             case CATEGORY_ITEM:
124.                 String categoryId = uri.getPathSegments().get(1);
125.                 updatedRows = db.update("Category", values, "id = ?", new String[]
126.
127.                                     { categoryId });
128.
129.                 break;
130.
131.             default:
132.                 break;
133.         }
134.         return updatedRows;
135.     }
136. }
137.
138.
139.
140.
141.
142.
143.
144.
145.
146.
147.
148.
149.
150.
151.
152.
153.
154.
155.
156.
157.
158.
159.
160.
161.
162.
163.
164.
165.
166.
167.
168.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204.
205.
206.
207.
208.
209.
210.
211.
212.
213.
214.
215.
216.
217.
218.
219.
220.
221.
222.
223.
224.
225.
226.
227.
228.
229.
230.
231.
232.
233.
234.
235.
236.
237.
238.
239.
240.
241.
242.
243.
244.
245.
246.
247.
248.
249.
250.
251.
252.
253.
254.
255.
256.
257.
258.
259.
260.
261.
262.
263.
264.
265.
266.
267.
268.
269.
270.
271.
272.
273.
274.
275.
276.
277.
278.
279.
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
290.
291.
292.
293.
294.
295.
296.
297.
298.
299.
300.
301.
302.
303.
304.
305.
306.
307.
308.
309.
310.
311.
312.
313.
314.
315.
316.
317.
318.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
330.
331.
332.
333.
334.
335.
336.
337.
338.
339.
340.
341.
342.
343.
344.
345.
346.
347.
348.
349.
350.
351.
352.
353.
354.
355.
356.
357.
358.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368.
369.
370.
371.
372.
373.
374.
375.
376.
377.
378.
379.
380.
381.
382.
383.
384.
385.
386.
387.
388.
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
400.
401.
402.
403.
404.
405.
406.
407.
408.
409.
410.
411.
412.
413.
414.
415.
416.
417.
418.
419.
420.
421.
422.
423.
424.
425.
426.
427.
428.
429.
430.
431.
432.
433.
434.
435.
436.
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
450.
451.
452.
453.
454.
455.
456.
457.
458.
459.
460.
461.
462.
463.
464.
465.
466.
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
480.
481.
482.
483.
484.
485.
486.
487.
488.
489.
490.
491.
492.
493.
494.
495.
496.
497.
498.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
510.
511.
512.
513.
514.
515.
516.
517.
518.
519.
520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539.
540.
541.
542.
543.
544.
545.
546.
547.
548.
549.
550.
551.
552.
553.
554.
555.
556.
557.
558.
559.
560.
561.
562.
563.
564.
565.
566.
567.
568.
569.
570.
571.
572.
573.
574.
575.
576.
577.
578.
579.
580.
581.
582.
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601.
602.
603.
604.
605.
606.
607.
608.
609.
610.
611.
612.
613.
614.
615.
616.
617.
618.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
630.
631.
632.
633.
634.
635.
636.
637.
638.
639.
640.
641.
642.
643.
644.
645.
646.
647.
648.
649.
650.
651.
652.
653.
654.
655.
656.
657.
658.
659.
660.
661.
662.
663.
664.
665.
666.
667.
668.
669.
670.
671.
672.
673.
674.
675.
676.
677.
678.
679.
680.
681.
682.
683.
684.
685.
686.
687.
688.
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
699.
700.
701.
702.
703.
704.
705.
706.
707.
708.
709.
710.
711.
712.
713.
714.
715.
716.
717.
718.
719.
720.
721.
722.
723.
724.
725.
726.
727.
728.
729.
730.
731.
732.
733.
734.
735.
736.
737.
738.
739.
740.
741.
742.
743.
744.
745.
746.
747.
748.
749.
750.
751.
752.
753.
754.
755.
756.
757.
758.
759.
760.
761.
762.
763.
764.
765.
766.
767.
768.
769.
770.
771.
772.
773.
774.
775.
776.
777.
778.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
789.
790.
791.
792.
793.
794.
795.
796.
797.
798.
799.
800.
801.
802.
803.
804.
805.
806.
807.
808.
809.
810.
811.
812.
813.
814.
815.
816.
817.
818.
819.
820.
821.
822.
823.
824.
825.
826.
827.
828.
829.
830.
831.
832.
833.
834.
835.
836.
837.
838.
839.
840.
841.
842.
843.
844.
845.
846.
847.
848.
849.
850.
851.
852.
853.
854.
855.
856.
857.
858.
859.
860.
861.
862.
863.
864.
865.
866.
867.
868.
869.
870.
871.
872.
873.
874.
875.
876.
877.
878.
879.
880.
881.
882.
883.
884.
885.
886.
887.
888.
889.
890.
891.
892.
893.
894.
895.
896.
897.
898.
899.
900.
901.
902.
903.
904.
905.
906.
907.
908.
909.
910.
911.
912.
913.
914.
915.
916.
917.
918.
919.
920.
921.
922.
923.
924.
925.
926.
927.
928.
929.
930.
931.
932.
933.
934.
935.
936.
937.
938.
939.
940.
941.
942.
943.
944.
945.
946.
947.
948.
949.
950.
951.
952.
953.
954.
955.
956.
957.
958.
959.
960.
961.
962.
963.
964.
965.
966.
967.
968.
969.
970.
971.
972.
973.
974.
975.
976.
977.
978.
979.
980.
981.
982.
983.
984.
985.
986.
987.
988.
989.
990.
991.
992.
993.
994.
995.
996.
997.
998.
999.
1000.

```

```
133.         @Override
134.         public int delete(Uri uri, String selection, String[] selectionArgs) {

135.             // 删除数据

136.             SQLiteDatabase db = dbHelper.getWritableDatabase();

137.             int deletedRows = 0;

138.             switch (uriMatcher.match(uri)) {
```

淘宝搜《[闵课通商学院](#)》，小白轻松拿高薪offer

```

139.
140.         case BOOK_DIR:
141.             deletedRows = db.delete("Book", selection, selectionArgs);
142.             break;
143.
144.         case BOOK_ITEM:
145.             String bookId = uri.getPathSegments().get(1);
146.             deletedRows = db.delete("Book", "id = ?", new String[] { bookId });
147.             break;
148.
149.         case CATEGORY_DIR:
150.             deletedRows = db.delete("Category", selection, selectionArgs);
151.             break;
152.
153.         case CATEGORY_ITEM:
154.             String categoryId = uri.getPathSegments().get(1);
155.             deletedRows = db.delete("Category", "id = ?", new String[]
156.                                     { categoryId });
157.             break;
158.
159.         default:
160.             break;
161.     }
162.

```

```

163.         return deletedRows;
164.     }
165.
166.     @Override
167.     public String getType(Uri uri) { 168.
169.         switch (uriMatcher.match(uri)) { 170.
171.             case BOOK_DIR:
172.                 return "vnd.android.cursor.dir/vnd.com.example.databasetest.
173.                     provider.book";
174.
175.             case BOOK_ITEM:
176.                 return "vnd.android.cursor.item/vnd.com.example.databasetest.
177.                     provider.book";
178.
179.             case CATEGORY_DIR:
180.                 return "vnd.android.cursor.dir/vnd.com.example.databasetest.

```

```

181.                provider.category";
182.
183.                case CATEGORY_ITEM:
184.                    return "vnd.android.cursor.item/vnd.com.example.databasetest.
185.                        provider.category";
186.
187.            }
188.
189.            return null;
190.
191.        }
192.    }
    
```

AndroidManifest.xml 文件中对内容提供者进行注册，标签为< provider >...< /provider >

```

1. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2.     package="com.example.databasetest"
3.     android:versionCode="1"
4.     android:versionName="1.0" >
5.     ...
6.     <application
7.         android:allowBackup="true"
8.         android:icon="@drawable/ic_launcher"
9.         android:label="@string/app_name"
10.        android:theme="@style/AppTheme" >
11.        ...
12.        <provider
13.            android:name="com.example.databasetest.DatabaseProvider"
14.
15.            android:authorities="com.example.databasetest.provider" >
16.
17.        </provider>
18.    </application>
19. </manifest>
    
```

这样一个完整的内容提供器的创建流程就讲完了。

## 4. ContentPrivoder 的原理机制

其实内容提供者可跨程序访问，这可以认为是一种进程间通信的方式，其实它原理核心就是 Binder。