**第七章 内容提供器**

**1.内容提供器简介**

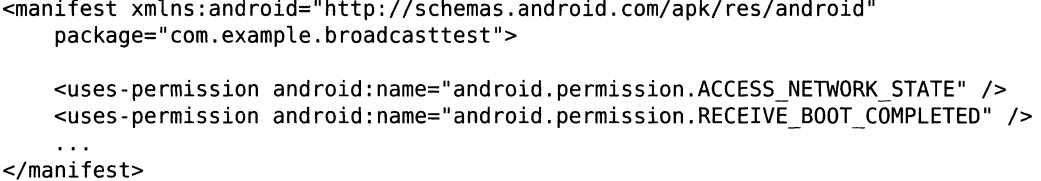
**内容提供器( Content Provider)主要用于在不同的应用程序之间实现数据共享的功能, 使用内容提供器是 Android实现跨程序共享数据的标准方式。**

**2.运行时权限**

**Android开发团队在 Android6.0系统中引用运行时权限, 用户不需要在安装软件的时候一次性授权所有申请的权限, 而是可以在软件的使用过程中再对某一项权限申请进行授权，从而更好地保护了用户的安全和隐私。**

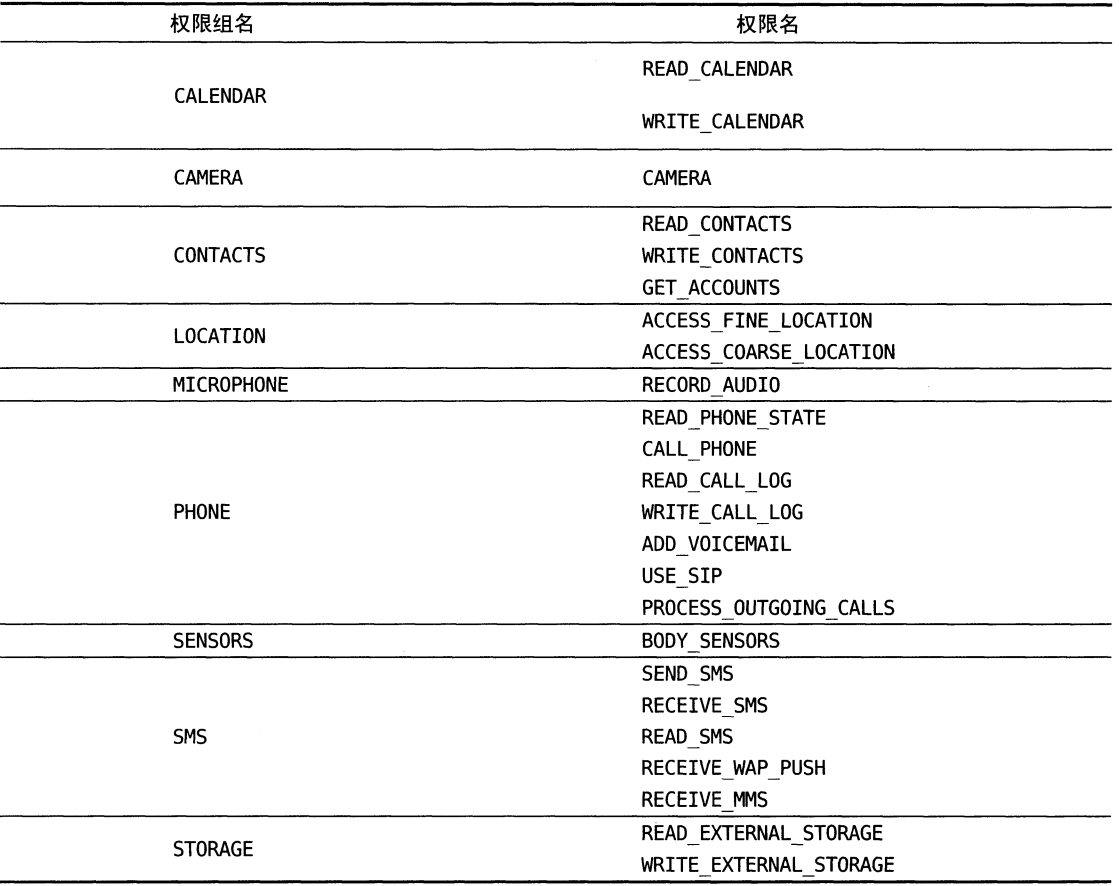
1. **Android权限机制**

**在第5章写 BroadcastTest项目时第一次接触了 Android权限相关的内容,当时为了要访同系统的网络状态以及监听开机广播, 于是在 AndroidManifest.xml文件中添加了这样两句权限声明:**

****

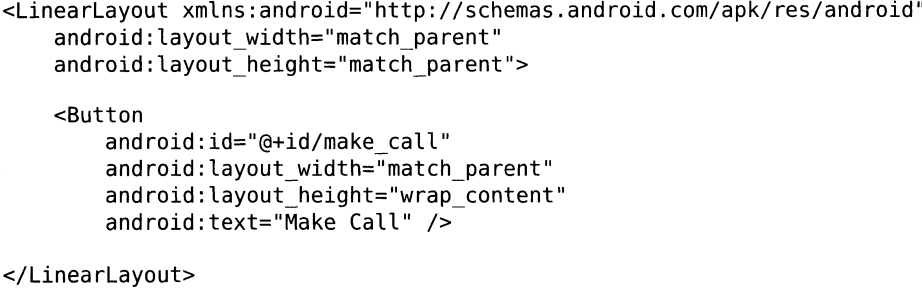
**这样用户主要在以下两个方面得到了保护, 一方面, 如果用户在低于6.0系统的设备上安装该程序，会在安装界面给出提醒。这样用户就可以清楚地知晓该程序一共申请了哪些权限, 从而决定是否要安装这个程序。 另一方面, 用户可以随时在应用程序管理界面査看任意一个程序的权限申请情况, 以此保证应用程序不出现各种滥用权限的情况。**

**Android将所有的权限归成了两类, 一类是普通权限, 一类是危险权限。下表另外除了所有危险权限一共9组24个：**

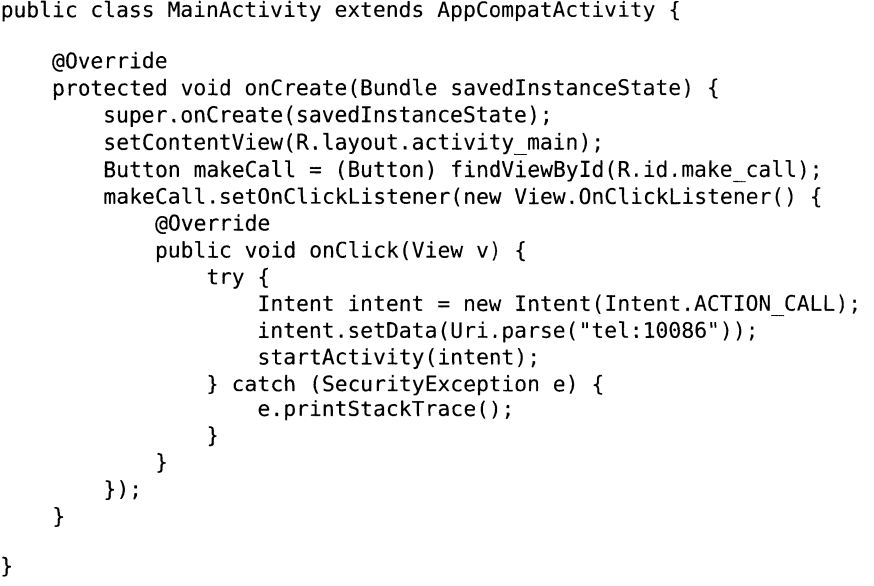


1. **在程序运行时申请权限**

**首先新建一个 RuntimePermissionTest项目,** **CALL\_PHONE这个权限是编写拨打电话功能的时候需要声明的, 因为拨打电活会涉及用户手机的资费问题,因而被列为了危险权限。修改 activity\_main.xm1布局文件, 如下所示:**

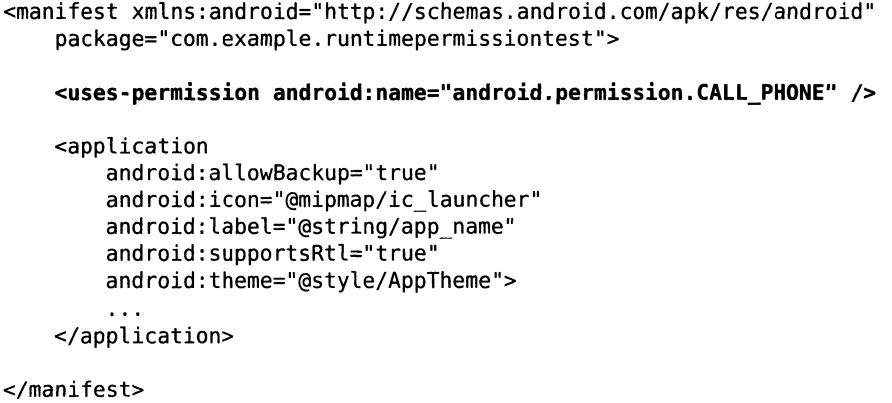


**接着修改MainActivity中的代码, 如下所示:**



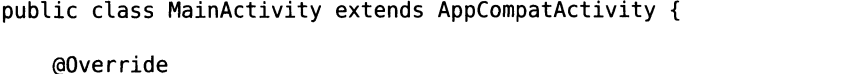
**在按钮的点击事件中,构建了一个隐式 Intent, Intent的 action指定为Intent.ACTION\_CALL,这是一个系统内置的打电话的动作。**

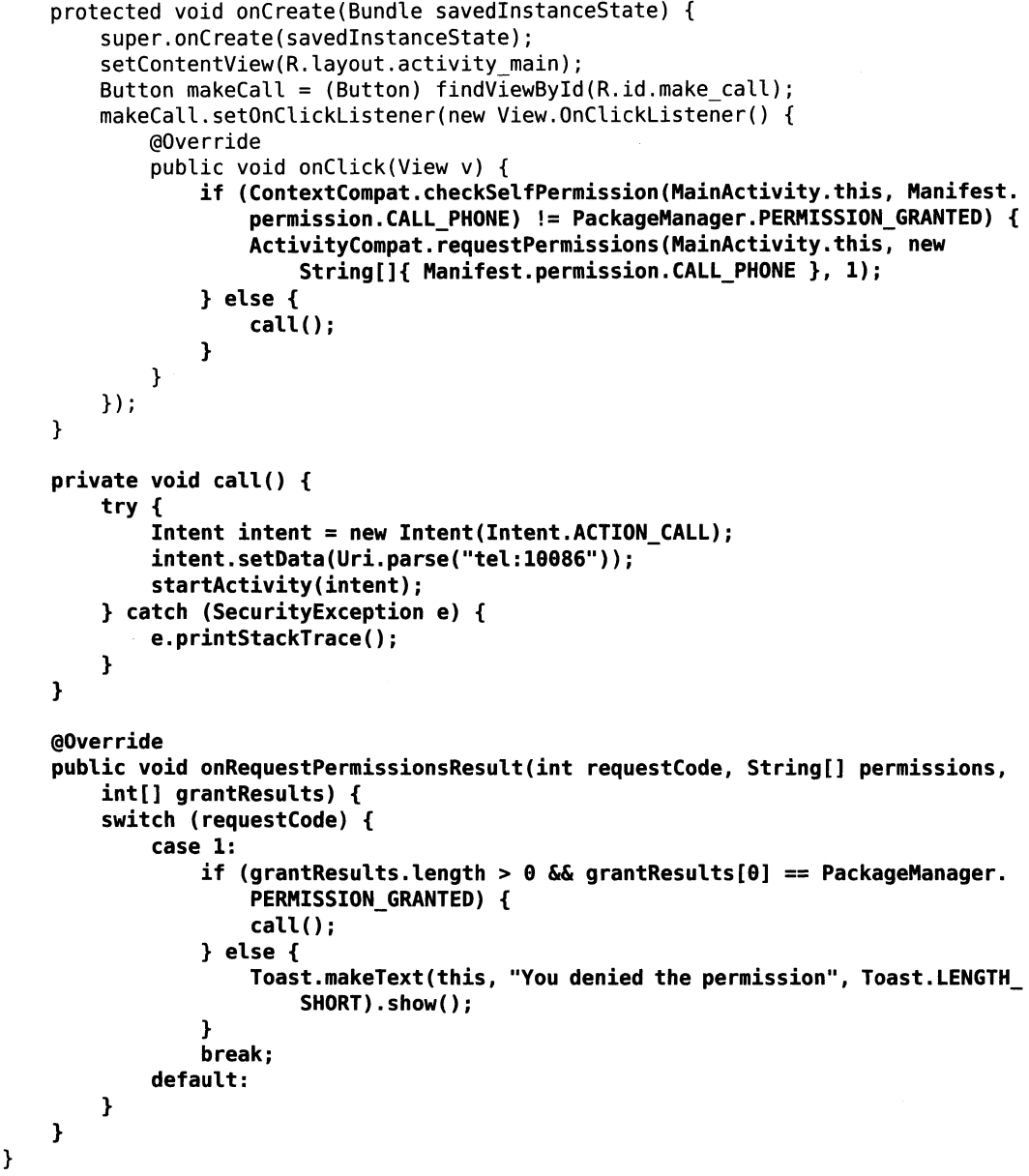
**那么接下来修改 AndroidManifest.xm1文件, 在其中声明如下权限:**



**这样就将拨打电话的功能成功实现了,并且在低于 Android6.0系统的手机上都是可以正常运行的,但是如果在6.0或者更高版本系统的手机上就不能运行。这由于权限被禁止所导致的,因为6.0及以上系统在使用危险权限时都必须进行运行时权限处理。**

**那么下面来尝试修复这个问题, 修改 MainActivity 中的代码, 如下所示:**





**运行时限的核心就是在程序运行过程中由用户授权我们去执行某些危险操作,程序是不可以擅自做主去执行这些危险操作。因此,第一步就是要先借助ContextCompat. checkSelfPermission( )方法判断用户是不是已经给过我们授权了。然后使用方法的返回值和 PackageManager.** **PERMISSION\_GRANTED做比较,相等就说明用户已经授权,不等就表示用户没有授权。 如果已经授权的话，直接去执行拨打电话的逻辑操作。如果没有授权的话, 则需要调用 Activitycompat.** **requestPermissions( )方法来向用户中请授权。**

**3 访问其他程序中的数据**

**内容提供器的用法一般有两种, 一种是使用现有的内容提供器来读取和操作相应程序中的数据, 另一种是创建自己的内容提供器给我们程序的数据提供外部访问接口。**

1. **ContentResolver的基本用法**

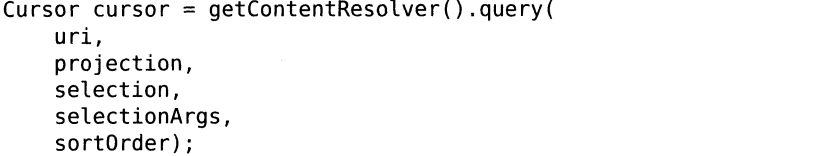
**对于每一个应用程序来说, 如果想要访同内容提供器中共享的数据, 就一定要借助 Content** **Resolver类,可以通过 Context中的 getContentResoLver()方法获取到该类的实例。 Content** **Resolver中提供了一系列的方法用于对数据进行 CRUD操作，不同于 SQLiteDatabase, ContentResolver中的增删改查方法都是不接收表名参数的, 而是使用一个 uri参数代替,这个参数被称为内容URI。**

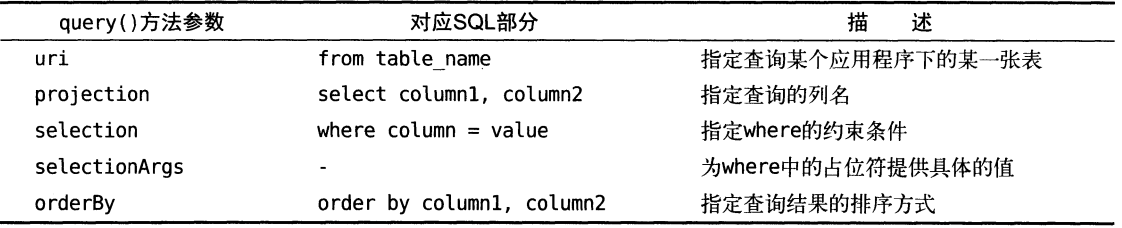
**内容URI主要由两部分组成: authority 和 path。authority是用于对不同的应用程序做区分的,** **path则是用于对同一应用程序中不同的表做区分的,通常都会添加到 authority的后面。**

**在得到了内容 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。代码如下所示：**

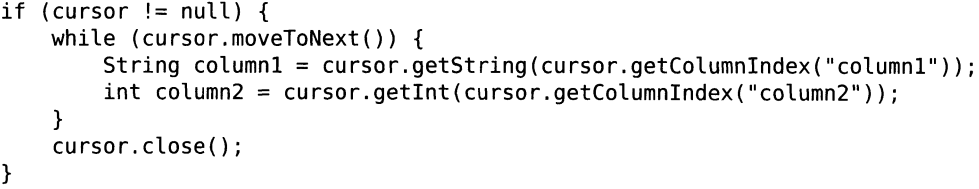


**使用Uri对象来査询 table1表中的数据, 代码如下所示:**

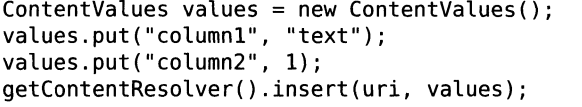




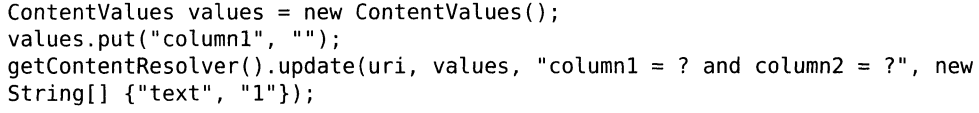
**查询完成后返回的仍然是一个 Cursor对象, 这时就可以将数据从 Cursor对象中逐个读取出来了 。 读取的思路仍然是通过移动游标的位置来遍历 Cursor的所有行, 然后再取出每一行中相应列的数据, 代码如下所示:**



**向table表中添加一条数据，代码如下所示：**



**想要更新这条新添加的数据,把 column1的值清空,可以借助 ContentResolver的 update()方法实现,代码如下所示:**

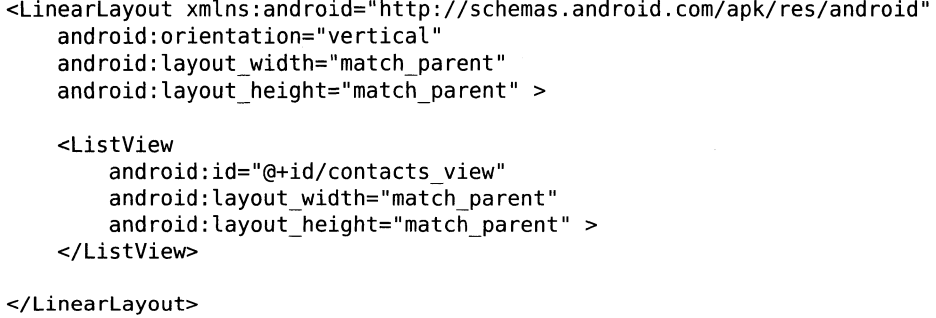


**用 ContentResolver的 delete( )方法将这条数据删除掉, 代码如下所示:**



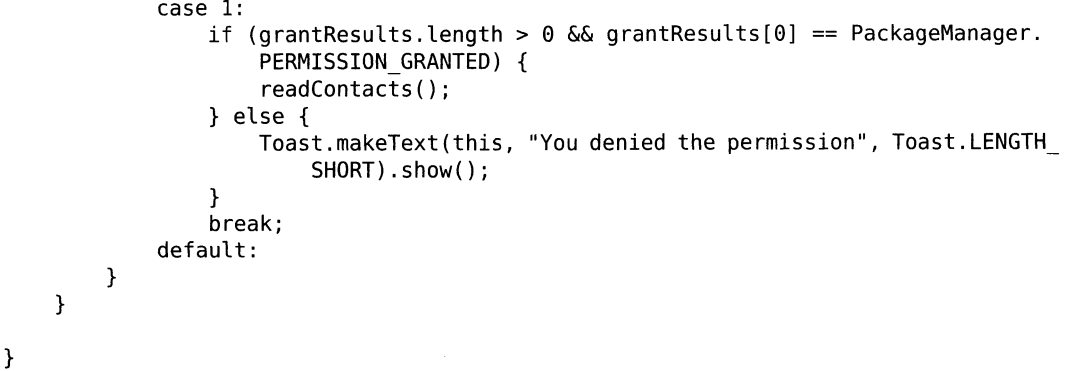
1. **读取系统联系人**

**新建一个 ContactsTest项目,** **编写一下布局文件,这里我们希望读取出来的联系人信息能够在 Listview中显示,因此,修改 activity\_main.xm1中的代码,如下所示:**

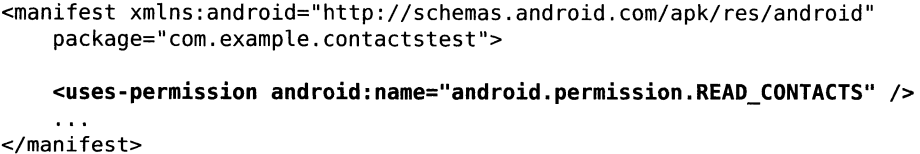


**接着修改 MainActivity 中的代码, 如下所示:**





**修改AndroidManifest.xm1中的代码,如下所示:**

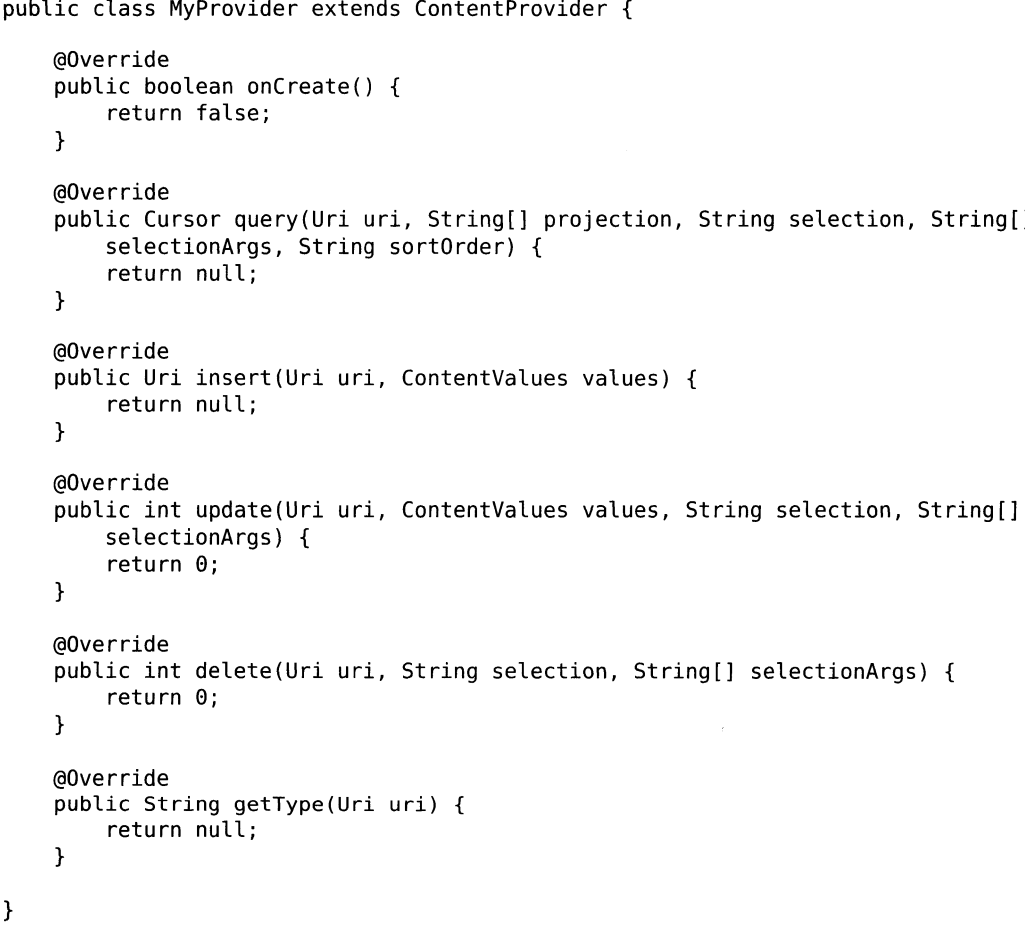


**加入了 android. Perm.ssion. READ\_CONTACTS权限, 这样此程序就可以访同到系统的联系人数据了。**

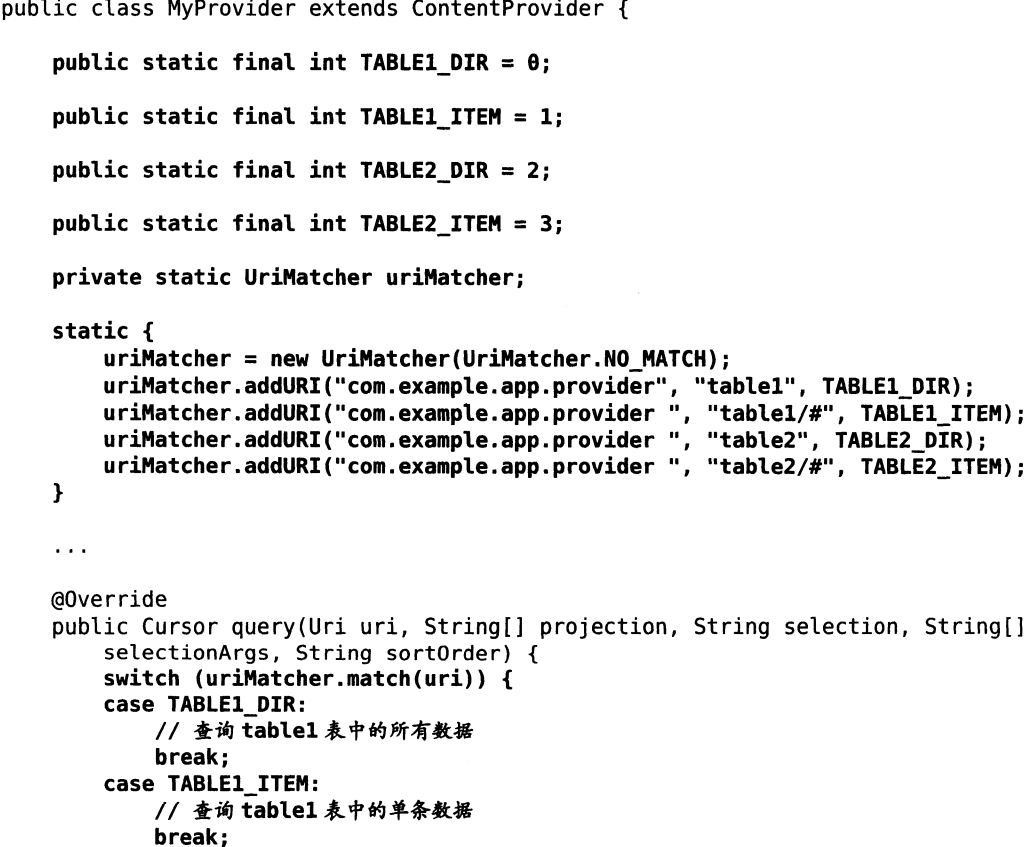
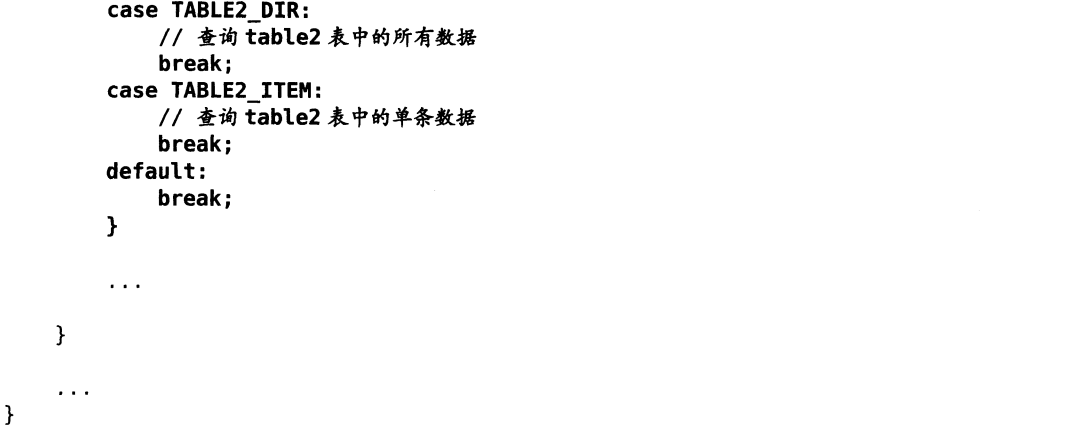
**4 创建自己的内容提供器**

**（1）创建内容提供器的步骤**

**要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器, 可以通过新建一个类去继承 ContentProvider的方式来创建一个自己的内容提供器。ContentProvider类中有6个抽象方法,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继承自 ContentProvider,代码如下所示:**



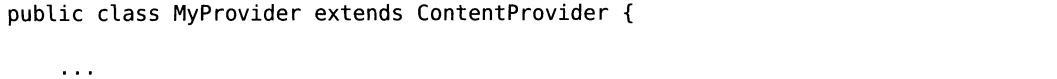
**借助 UriMatcher类就可以轻松地实现匹配内容 URI 的功能。 UriMatcher中提供了一个 adduRI()方法,这个方法接收3个参数,可以分别把 authority、 path和一个自定义代码传进去。修改 MyProvider中的代码, 如下所示:**

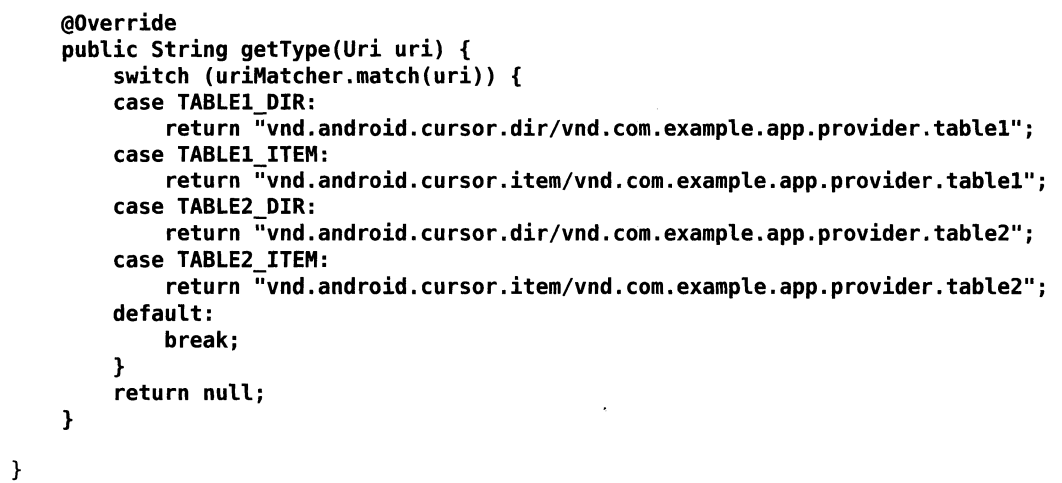


**MyProvider中新增了4个整型常量,其中 TABLE1\_DIR表示访问 table1表中的所有数据, TABLE1\_ITEM表不访问 table1表中的单条数据, TABLE2\_DIR表示访同 table2表中的所有数据, TABLE2\_ITEM表示访问 table2表中的单条数据。接着在静态代码块里我们创建了UriMatcher的实例,并调用 addU RI( )方法,将期望匹配的内容 URL格式传递进去。然后当query（）方法被调用的时候，就会通过UriMatcher的match（）方法对传入的Uri对象进行匹配，如果发现UriMatcher中某个内容URI格式成功匹配了该Uri对象，则会返回相应的自定义代码，然后就可以判断出调用方期望访问的到底是什么数据了。**

**getType( )方法是所有的内容提供器都必须提供的一个方法,用于获取 uri对象所对应的 MIME类型。**

**实现 getType()方法中的逻辑,代码如下所示:**

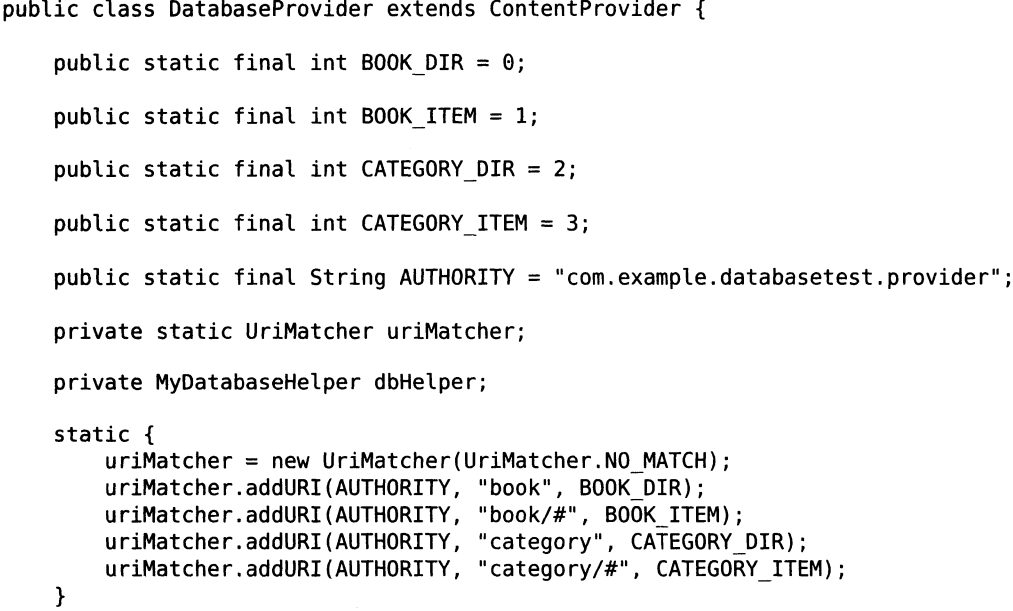


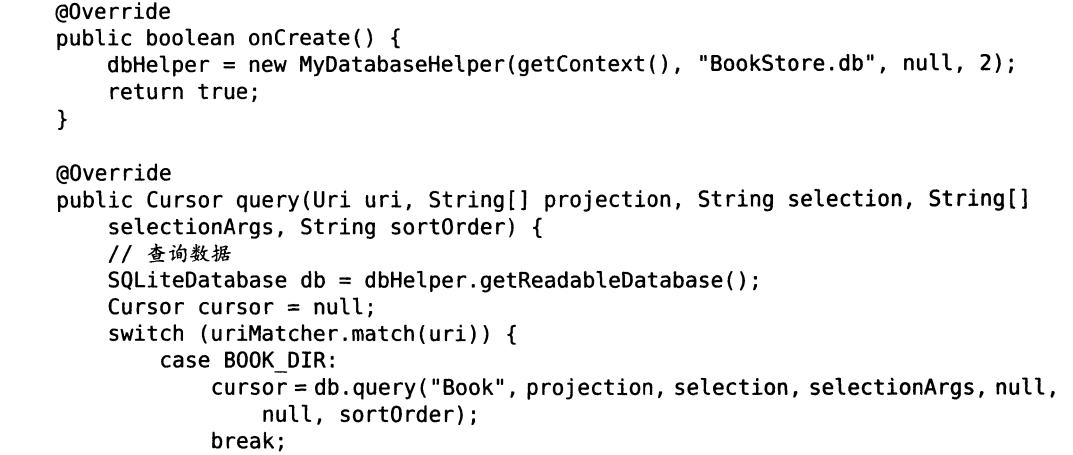


**到这里, 一个完整的内容提供器就创建完成了, 现在任何一个应用程序都可以使用ContentResolver来访问我们程序中的数据。**

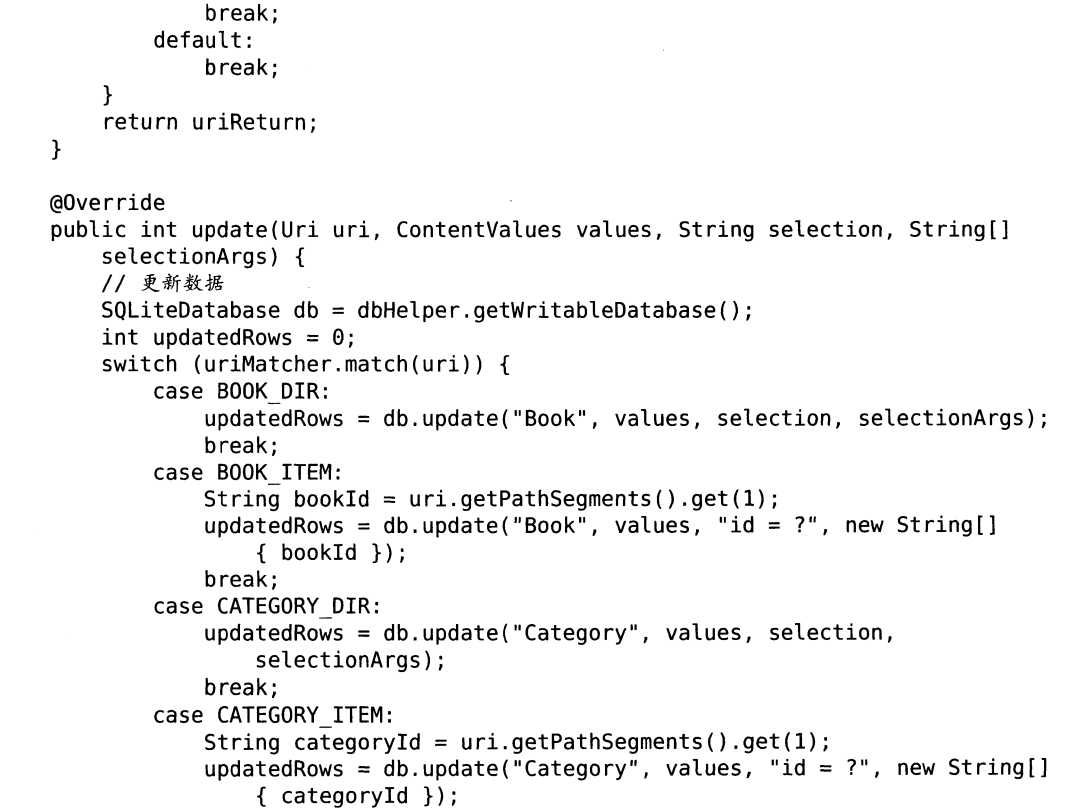
1. **实现跨程序数据共享**

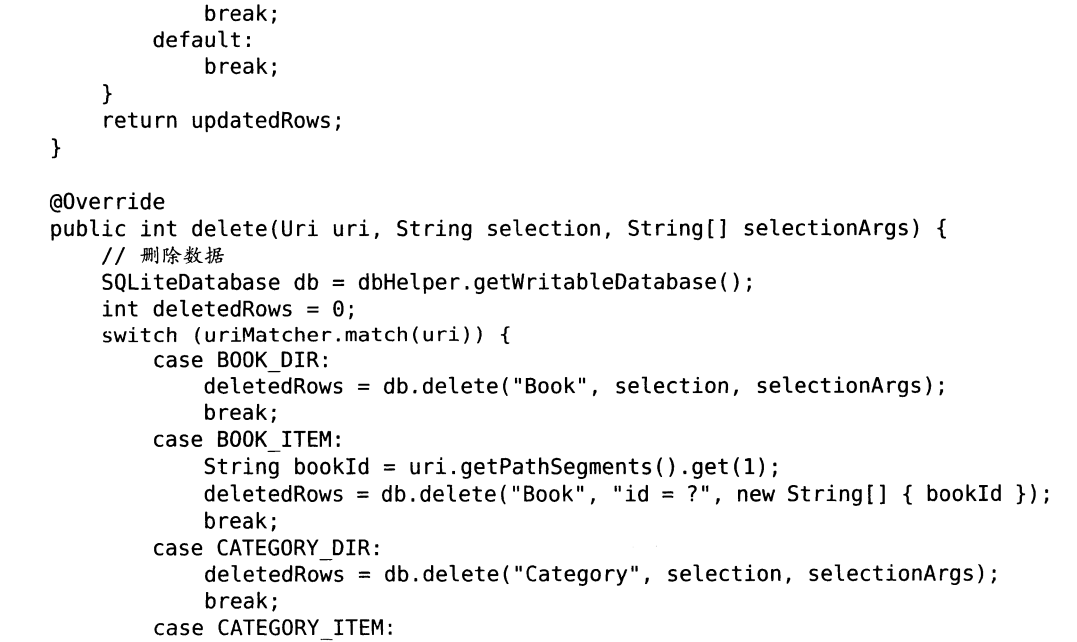
**打开 DatabaseTest项目，创建一个内容提供器,右击 com.example.broadcasttest包→New→0ther→Content Provider点击 Finish完成创建。修改 DatabaseProvider中的代码, 如下所示：**

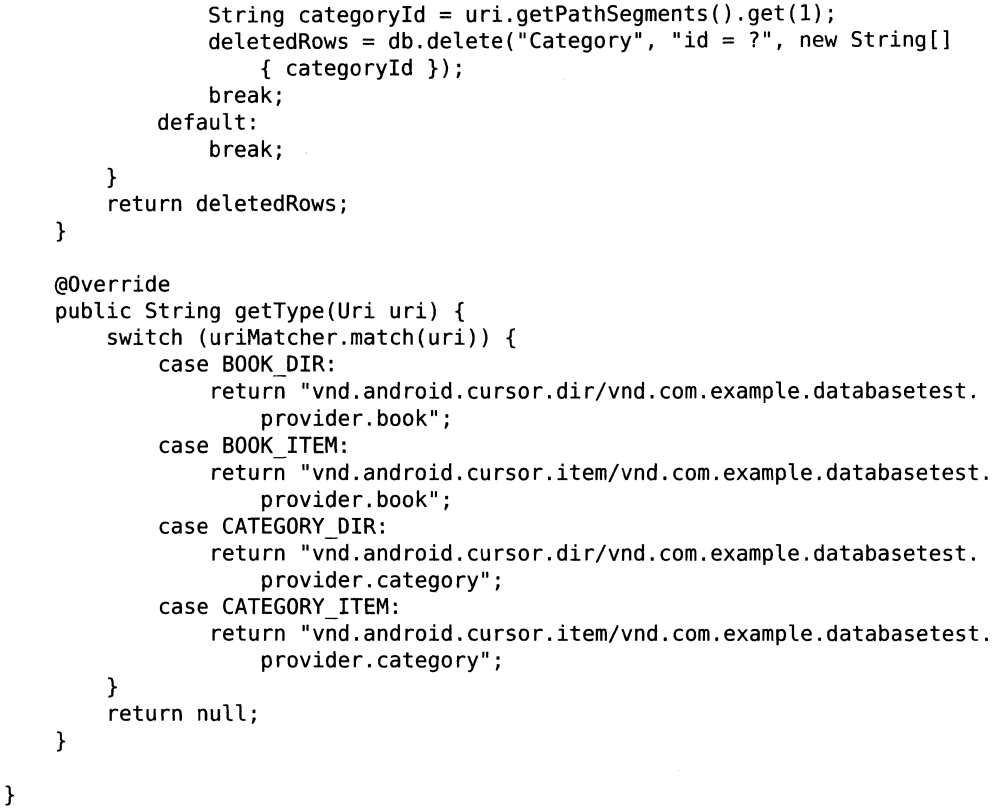












**首先在类的一开始,同样是定义了4个常量,分别用于表示访同 Book表中的所有数据、访同 Book表中的单条数据、访间 Category 表中的所有数据和访问 category 表中的单条数据。然后在静态代码块里对 UriMatcher进行了初始化操作,将期望匹配的几种 URI 格式添加了进去。 接下来就是每个抽象方法的具体实现：OnCreate()方法，query( )方法, insert( )方法,** **update( )方法，delete()方法，getType( )方法。**

**内容提供器一定要在 AndroidManifest.xm1文件中注册才可以使用。打开 AndroidManifest.xm1文件**

**代码如下所示：**



