1. 找入口

每个应用都是通过应用列表进入相应程序，那么如何判断入口呢？

（看 “应用入口”文件夹进行讲解）

结论：

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category ndroid:name=*"android.intent.category.LAUNCHER"* />

</intent-filter>

那么是否可以对多个Activity设置该intent-filter呢？

答案是根据实际需要，可以这么做，但要在代码中做相应处理。

|  |  |  |
| --- | --- | --- |
| ACTION\_MAIN | android.intent.action.MAIN | 应用程序入口 |
|  |  |  |
|  |  |  |
| ACTION\_DIAL | android.intent.action.DIAL | 显示拨号面板 |
| ACTION\_CALL | android.intent.action.CALL | 直接向指定用户打电话 |
| ACTION\_SEND | android.intent.action.SEND | 向其他人发送数据 |
| ACTION\_SENDTO | android.intent.action.SENDTO | 向其他人发送消息 |
| CATEGORY\_LAUNCHER | android.intent.category.LAUNCHER | Activity显示顶级程序列表中 |
| CATEGORY\_DEFAULT | android.intent.category.DEFAULT | 默认的Category |
| CATEGORY\_HOME | android.intent.category.HOME | 设置该Activity随系统启动而运行 |
|  |  |  |

讲解Launcher时遇到 intent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

FLAG\_ACTIVITY\_NEW\_TASK: 设置此状态，记住以下原则，首先会查找是否存在和被启动的Activity具有相同的亲和性的任务栈（即taskAffinity，注意同一个应用程序中的activity的亲和性一样，所以下面的a情况会在同一个栈中，前面这句话有点拗口，请多读几遍），如果有，刚直接把这个栈整体移动到前台，并保持栈中的状态不变，即栈中的activity顺序不变，如果没有，则新建一个栈来存放被启动的activity

         a. 前提: Activity A和Activity B在同一个应用中.

             操作: Activity A启动，开辟Task堆栈(堆栈状态: A), 在Activity A中启动Activity B, 启动Activity B的Intent的Flag设为

                     FLAG\_ACTIVITY\_NEW\_TASK, Activity B被压入Activity A所在堆栈(堆栈状态: AB).

            原因: 默认情况下同一个应用中的所有Activity拥有相同的关系(taskAffinity).

         b. 前提: Activity A在名称为"TaskOne应用"的应用中, Activity C和Activity D在名称为"TaskTwo应用"的应用中.

             操作1: 在Launcher中单击"TaskOne应用"图标, Activity A启动开辟Task堆栈, 命名为TaskA(TaskA堆栈状态: A),

                       在Activity A中启动Activity C, 启动Activity C的Intent的Flag设为FLAG\_ACTIVITY\_NEW\_TASK,

                       Android系统会为Activity C开辟一个新的Task, 命名为TaskB(TaskB堆栈状态: C), 长按Home键, 选择TaskA,

                       Activity A回到前台, 再次启动Activity C（两种情况1.从桌面启动；2.从Activity A启动，两种情况一样）, 这时TaskB回到前台, Activity C显示, 供用户使用, 即:

                       包含FLAG\_ACTIVITY\_NEW\_TASK的Intent启动Activity的Task正在运行, 则不会为该Activity创建新的Task,

                       而是将原有的Task返回到前台显示.

             操作2: 在Launcher中单击"TaskOne应用"图标, Activity A启动开辟Task堆栈, 命名为TaskA(TaskA堆栈状态: A),

                       在Activity A中启动Activity C,启动Activity C的Intent的Flag设为FLAG\_ACTIVITY\_NEW\_TASK,

                       Android系统会为Activity C开辟一个新的Task, 命名为TaskB(TaskB堆栈状态: C),  在Activity C中启动

                       Activity D(TaskB的状态: CD) 长按Home键, 选择TaskA, Activity A回到前台, 再次启动Activity C(从桌面或者ActivityA启动，也是一样的),

                       这时TaskB回到前台, Activity D显示,供用户使用.说明了在此种情况下设置FLAG\_ACTIVITY\_NEW\_TASK后，会先查找是不是有Activity C存在的栈，根据亲和　　　　　　　　　　　　　性(taskAffinity)，如果有，刚直接把这个栈整体移动到前台，并保持栈中的状态不变，即栈中的顺序不变

SingleTop：如果某个Activity的Launch mode设置成singleTop，那么当该Activity位于栈顶的时候，再通过Intent跳转到本身这个Activity，则将不会创建一个新的实例压入栈中。例如：现在栈的情况为：A B C D。D的Launch mode设置成了singleTop，那么在D中启动Intent跳转到D，那么将不会新创建一个D的实例压入栈中，此时栈的情况依然为：A B C D。但是如果此时B的模式也是singleTop，D跳转到B，那么则会新建一个B的实例压入栈中，因为此时B不是位于栈顶，此时栈的情况就变成了：A B C D B。

Activity生命周期（见图Activity生命周期）

大家注意到，除了几个常见的方法外，我们还添加了onWindowFocusChanged、onSaveInstanceState、onRestoreInstanceState方法：

1. onWindowFocusChanged方法：在Activity窗口获得或失去焦点时被调用，例如创建时首次呈现在用户面前；当前Activity被其他Activity覆盖；当前Activity转到其他Activity或按Home键回到主屏，自身退居后台；用户退出当前Activity。以上几种情况都会调用onWindowFocusChanged，并且当Activity被创建时是在onResume之后被调用，当Activity被覆盖或者退居后台或者当前Activity退出时，它是在onPause之后被调用。（见图onWindowFocusChanged调用顺序）。
2. onSaveInstanceState：(1)在Activity被覆盖或退居后台之后，系统资源不足将其杀死，此方法会被调用；(2)在用户改变屏幕方向时，此方法会被调用；(3)在当前Activity跳转到其他Activity或者按Home键回到主屏，自身退居后台时，此方法会被调用。第一种情况我们无法保证什么时候发生，系统根据资源紧张程度去调度；第二种是屏幕翻转方向时，系统先销毁当前的Activity，然后再重建一个新的，调用此方法时，我们可以保存一些临时数据；第三种情况系统调用此方法是为了保存当前窗口各个View组件的状态。onSaveInstanceState的调用顺序是在onPause之前。（见图onSave\_RestoreInstanceState调用顺序）
3. onRestoreInstanceState：(1)在Activity被覆盖或退居后台之后，系统资源不足将其杀死，然后用户又回到了此Activity，此方法会被调用；(2)在用户改变屏幕方向时，重建的过程中，此方法会被调用。我们可以重写此方法，以便可以恢复一些临时数据。onRestoreInstanceState的调用顺序是在onStart之后。（见图onSave\_RestoreInstanceState调用顺序）

提一下class BackgroundQueryHandler extends AsyncQueryHandler，暂时只需了解AsyncQueryHandler异步处理，你给个query它带回一个结果。

 ListView是Android开发过程中较为常见的组件之一，它将数据以列表的形式展现出来。一般而言，一个ListView由以下三个元素组成：

       1、View，用于展示列表，通常是一个xml所指定的。大家都知道Android的界面基本上是由xml文件负责完成的，所以ListView的界面也理所应当的使用了xml定义。例如在ListView中经常用到的“android.R.layout.simple\_list\_item”等， 就是Android系统内部定义好的一个xml文件。

2、适配器，用来将不同的数据映射到View上。不同的数据对应不同的适配器，如BaseAdapter、ArrayAdapter、CursorAdapter、SimpleAdapter等， 他们能够将数组、指针指向的数据、Map等数据映射到View上。也正是由于适配器的存在，使得ListView的使用相当灵活，经过适配器的处理后，在 view看来所有的数据映射过来都是一样的。

       3、数据，具体的来映射数据和资源，可以是字符串，图片等。通过适配器，这些数据将会被实现到 ListView上。所有的数据和资源要显示到ListView上都通过适配器来完成。

（见图ListView与Adapter关系）

系统已有的适配器可以将基本的数据显示到ListView上，如：数组，Cursor指向的数据，Map里的数据。但是在实际开发中这些系统已实现的适配器，有时不能满足我们的需求。而且系统自带的含有多选功能ListView在实际使用过程中会有一些问题。要实现复杂的ListView可以通过继承ListView并重写相应的方法完成，同时也可以通过继承BaseAdapter来实现。

在这里，“public class NotesListAdapter extends CursorAdapter ”，而CursorAdapter继承自BaseAdapter，由于CursorAdapter已经实现BaseAdapter的绝大部分方法（见图CursorAdapter的getView实现、CursorAdapter关于BaseAdapter的其它实现），你只需要实现newView和bindView就能将数据绑定到ListView的Item上去。

至此，我们了解了ListView与Adapter之间处理关系。在此之前，大家可能会对BackgroundQueryHandler查询结果的来源感兴趣。

SQLiteOpenHelper使用详解

Android平台提供给我们一个数据库辅助类来创建或打开数据库，这个辅助类继承自SQLiteOpenHelper类

在该类的构造器中，调用Context中的方法创建并打开一个指定名称的数据库对象。继承和扩展SQLiteOpenHelper类主要做的工作就是重写以下两个方法。

onCreate(SQLiteDatabase db) : 当数据库被首次创建时执行该方法，一般将创建表等初始化操作在该方法中执行。（见图NotesDatabaseHelper的onCreate实现）

onUpgrade(SQLiteDatabse dv, int oldVersion,int new Version)：当打开数据库时传入的版本号与当前的版本号不同时会调用该方法。（见图NotesDatabaseHelper的onUpgrade实现）

还有一个方法选用：

onOpen() ：当每次打开数据库时被调用。

注意 ：最后加入对SD卡的读写权限：

|  |  |
| --- | --- |
| 1 | <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"></uses-permission> |

我们注意到，这只是实现了表的创建，和我们前边的查询、插入、删除等操作，似乎没有关系。别急，下面就是介绍NotesProvider的实现。

ContentProvider在android中的作用是对外共享数据，也就是说你可以通过ContentProvider把应用中的数据共享给其他应用访问，其他应用可以通过ContentProvider对你应用中的数据进行添删改查。这里为何要使用ContentProvider对外共享数据呢？是这样的，如果采用文件操作模式对外共享数据，数据的访问方式会因数据存储的方式而不同，导致数据的访问方式无法统一，如：采用xml文件对外共享数据，需要进行xml解析才能读取数据；采用sharedpreferences共享数据，需要使用sharedpreferences API读取数据。  
使用ContentProvider对外共享数据的好处是统一了数据的访问方式。

通过ContentProvider对外共享数据时，第一步需要继承ContentProvider并重写下面方法（以NoteProvider为例）：（见图NoteProvider要实现的方法）

第二步需要在AndroidManifest.xml使用<provider>对该ContentProvider进行配置，为了能让其他应用找到该ContentProvider ，ContentProvider采用了authorities（主机名/域名）对它进行唯一标识，你可以把ContentProvider看作是一个网站（想想，网站也是提供数据者），authorities 就是他的域名：（见图NoteProvider在manifest中配置）

Uri代表了要操作的数据，Uri主要包含了两部分信息：1》需要操作的ContentProvider ，2》对ContentProvider中的什么数据进行操作，一个Uri由以下几部分组成：

<scheme>://<authority><path>?<query>

ContentProvider的scheme已经由Android所规定，scheme为content://。   
主机名（authorities）用于唯一标识这个ContentProvider，外部调用者可以根据  
它找到对应的内容提供者（ContentProvider）。   
路径（Path）可以用来表示我们要操作的数据，路径的构建应该根据业务而定。

以NoteProvider为例，当query中的Uri为content://micode\_notes/note/2时，实际就是查询note表中的ID为2的记录 。

UriMatcher类使用介绍

因为Uri代表了要操作的数据，所以我们经常需要解析Uri，并从Uri中获取数  
据。Android提供了两个用于操作Uri的工具类，分别为UriMatcher和ContentUris。   
UriMatcher类用于匹配Uri，用法如下：

1. 首先需要把匹配的Uri路径全部进行注册，如下（以本项目为例）： （见图UriMatcher实例）
2. 注册完需要匹配的Uri后，就可以使用sMatcher.match(uri)对输入的uri进  
   匹配，如果匹配正确就返回匹配码，匹配码是addUri()方法传入的第三个  
   数，假设匹配content://micode\_notes/note/2 路径，返回  
   匹配码为URI\_NOTE\_ITEM。

使用ContentResovler操作ContentProvider中的数据

当外部应用需要对ContentProvider 中的数据进行操作时，可以使用  
ContentResovler来完成，可以使用Activity的getContentResovler()方法来获取  
CountentResovler对象。ContentResovler提过了和ContentProvider类似的四个  
方法，这里不再赘述。

最后获得的Cursor（见图Cursor获取）就是此前Adapter获得数据的来源（见图mNotesListAdapter中Cursor来源）