宝贵建议请发送至：wangzhenyang@itcast.cn

黑马程序员 itheima.com -编程，始于黑马

Android 课程同步笔记

Alpha 0.01 版

By 阳哥

Android-ContentProvider

1.ContentProvider 简介（★★★）

ContentProvider 即内容提供者，是 Android 的四大组件之一。内容提供者是应用程序之间共享数据的接口，Android 系统将这种机制应用到方方面面。比如：联系人 Provider 专为不同应用程序提供联系人数据；设置 Provider 专为不同应用程序提供系统配置信息，包括内置的设置应用程序等。当应用继承 ContentProvider 类，并重写该类用于提供数据和存储数据的方法，就可以向其他应用共享其数据。虽然使用其他方法也可以对外共享数据，但数据访问方式会因数据存储的方式而不同，如：采用文件方式对外共享数据，需要进行文件操作读写数据；采用 SharedPreferences 共享数据，需要使用 SharedPreferences API 读写数据。而使用 ContentProvider 共享数据的好处是统一了数据访问方式。

1.1 创建一个 ContentProvider

Tips：为了方便演示 ContentProvider 的使用，我们下面将编写一个案例，跟着案例一点点学习 ContentProvider 的用法，相信当大家能独自把该案例写出来的时候也是我们基本掌握 ContentProvider 使用方法的时候。

需求：在本应用下创建一个 person 数据库，对外提供一个 ContentProvider 接口，使外部程序可以通过我们提供的 ContentProvider 接口对我们的 person 数据库进行增删改查操作。

1 新创建一个 Android 工程，工程名《MyContentProvider》，包名：com.itheima.provider。

2 在 com.itheima.provider.dao 包下新建一个 PersonOpenHelper 类继承 SQLiteOpenHelper 类，该类用于创建数据库。

```
public class PersonOpenHelper extends SQLiteOpenHelper {

    public PersonOpenHelper(Context context, String name,
CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    /**
     * 对外提供一个简单的构造函数，使用默认的数据库和默认的版本号
     * @param context
     */
    public PersonOpenHelper(Context context){
        super(context, "person.db", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "create table person (id integer primary key
autoincrement,name varchar(20),phone varchar(20),age integer,address
varchar(50));";
        db.execSQL(sql);

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {

    }

}
```

3 在 com.itheima.contenProvider.provider 包中创建 ,PersonContentProvider 类继承 ContentProvider 类。同时将该 Provider 在 AndroidManifest.xml 中注册。

```
<provider
android:exported="true"
android:name="com.itheima.contenProvider.provider.PersonContentPro
vider"android:authorities="com.itheima.person" />
```

PersonContentProvider 类是核心业务代码，也是本文档的重要内容，代码清单如下：

```
public class PersonContentProvider extends ContentProvider {
    //用于存放并匹配个 Uri 标识信息，一般在静态代码块中对其信息进行初始化操
作
    private static UriMatcher matcher;
    //声明一个用于操作数据库对象
    private PersonOpenHelper openHelper;
    //主机名信息：对应清单文件的 authorities 属性
    private static final String AUTHORITY = "com.itheima.person";
    //数据库 表名
    private static final String TABLE_PERSON_NAME = "person";
    //Uri 匹配成功的返回码
    private static final int PERSON_INSERT_CODE = 1000;
    private static final int PERSON_DELETE_CODE = 10001;
    private static final int PERSON_UPDATE_CODE = 10002;
    private static final int PERSON_QUERYALL_CODE = 10003;
    private static final int PERSON_QUERYONE_CODE = 10004;
    //静态代码块，用于初始化 UriMatcher
    static{
        //NO_MATCH:没有 Uri 匹配的时候返回的状态码（-1）
        matcher = new UriMatcher(UriMatcher.NO_MATCH);
        //添加一个分机号：
        //对 person 表进行添加操作，如果
Uri=content://com.itheima.person/person/insert,则返回
PERSON_INSERT_CODE
        matcher.addURI(AUTHORITY, "person/insert",
PERSON_INSERT_CODE);
        //对 person 表进行删除操作,如果 Uri=
content://com.itheima.person/person/delete,则返回 PERSON_DELETE_CODE
        matcher.addURI(AUTHORITY, "person/delete",
PERSON_DELETE_CODE);
        //对 person 表进行修改操作,如果 Uri=
content://com.itheima.person/person/update,则返回 PERSON_UPDATE_CODE
        matcher.addURI(AUTHORITY, "person/update",
PERSON_UPDATE_CODE);
```

```
//对 person 表进行查询所有操作,如果 Uri=
content://com.itheima.person/person,则返回 PERSON_QUERYALL_CODE
        matcher.addURI(AUTHORITY, "person", PERSON_QUERYALL_CODE);
        //对 person 表进行查询单个操作,如果 Uri=
content://com.itheima.person/person/#,(#: 代表数字)则返回
PERSON_QUERYONE_CODE
        matcher.addURI(AUTHORITY, "person/#", PERSON_QUERYONE_CODE);
    }
    @Override
    public boolean onCreate() {
        openHelper = new PersonOpenHelper(getContext());
        return false;
    }
    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {
        //用匹配器去匹配 uri，如果匹配成功则返回匹配器中对应的状态码
        int matchCode = matcher.match(uri);
        SQLiteDatabase db = openHelper.getReadableDatabase();
        switch (matchCode) {
        case PERSON_QUERYALL_CODE:
            return db.query(TABLE_PERSON_NAME, projection, selection,
selectionArgs, null, null, sortOrder);
        case PERSON_QUERYONE_CODE:
        //使用 ContentUris 工具类解析出 uri 中的 id
            long parseId = ContentUris.parseId(uri);
            return db.query(TABLE_PERSON_NAME, projection,"id=?", new
String[]{parseId+""}, null, null, sortOrder);
        default:
            throw new IllegalArgumentException("Uri 匹配失败: "+uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        SQLiteDatabase db = openHelper.getWritableDatabase();
        //新插入对象的 id
        long id = db.insert(TABLE_PERSON_NAME, null, values);
        db.close();
        //使用 ContentUris 工具类将 id 追加到 uri 中，返回给客户
        return ContentUris.withAppendedId(uri, id);
```

```
    }

    @Override
    public int delete(Uri uri, String selection, String[]
selectionArgs) {
        SQLiteDatabase db = openHelper.getWritableDatabase();
        //返回删除的个数
        int count = db.delete(TABLE_PERSON_NAME, selection,
selectionArgs);
        //关闭数据库
        db.close();
        return count;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
String[] selectionArgs) {
        SQLiteDatabase db = openHelper.getWritableDatabase();
        //返回更新的个数
        int count = db.update(TABLE_PERSON_NAME, values, selection,
selectionArgs);
        //更新数据库
        db.close();
        return count;
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }

}
```

1.2 访问 ContentProvider

外部程序只需知道内容提供者的 Uri 路径信息，通过 ContentResolver 即可调用内容提供者。

Uri 的组成：

- schema，用来说明一个 ContentProvider 控制这些数据。 "content://"
- 主机名或授权 Authority：它定义了是哪个 ContentProvider 提供这些数据。
- path：路径，URI 下的某一个 Item。
- ID：通常定义 Uri 时使用”#”号占位符代替，使用时替换成对应的数字
- content://com.itheima.provider/person/#：#表示数据 id（#代表任意数字）

  content://com.itheima.provider/person/*：*来匹配任意文本。

Tips：为了演示如何调用 ContentProvider，我们新建一个 Android 测试工程，访问本文档 1.1 章节中提供的数据。

1 新创建一个 Android 工程，工程名《TestContentProvider》，包名：com.itheima.contentProviderTest。

2 在 AndroidManifest.xml 中添加 `instrumentation` 和 `uses-library`。

```
<instrumentation
android:name="android.test.InstrumentationTestRunner"
android:targetPackage="com.itheima.contextProviderTest"/>
```

```
<uses-library android:name="android.test.runner"/>
```

Tips:其中 `instrumentation` 放在清单文件的 `manifest` 节点下，`uses-library` 放在 `application` 节点下。

3 在 com.itheima.contentProviderTest 包下，新建 TestContentProvider 类，在该类中实现核心方法，调用 1.1 章节中 ContentProvider。

- 测试 insert 方法

```
public void insertTest(){
      ContentResolver resolver = getContext().getContentResolver();
      Uri uri =
Uri.parse("content://com.itheima.person/person/insert");
      ContentValues values = new ContentValues();
      values.put("name", "张三");
      values.put("age", 22);
      values.put("phone", "13240217777");
      values.put("address", "北京");

      Uri insert = resolver.insert(uri, values);
      System.out.println(insert);
      long id = ContentUris.parseId(insert);
      System.out.println("插入数据产生的 id 是: "+id);
   }
```

Tips：在我第一次做测试时报了如下图表中的异常，意思就是我没有权限访问

com.itheima.contenProvider 包中提供的 ContentProvider，原因是我忘记了在 1.1 章节

中 AndroidManifest.xml 文件中给 provider 标签添加 android:exported="true"

属性，该属性的意思是是否对外发布，系统默认为 false，这里一定记住要设置为 true。

```
java.lang.SecurityException: Permission Denial: opening provider
com.itheima.contenProvider.provider.PersonContentProvider from
ProcessRecord{b610aaa8
2082:com.itheima.contextProviderTest/u0a10090} (pid=2082, uid=10090)
that is not exported from uid 10091
at android.os.Parcel.readException(Parcel.java:1425)
at android.os.Parcel.readException(Parcel.java:1379)
at
android.app.ActivityManagerProxy.getContentProvider(ActivityManage
rNative.java:2530)
at
android.app.ActivityThread.acquireProvider(ActivityThread.java:446
```

- 测试 update 方法

```
public void updateTest(){
      ContentResolver resolver = getContext().getContentResolver();
      Uri uri =
Uri.parse("content://com.itheima.person/person/update");
      ContentValues values = new ContentValues();
      values.put("name", "李四");
      int count = resolver.update(uri, values , "name=?", new
String[]{"张三"});
      System.out.println("更新的个数为: "+count);
   }
```

测试 delete 方法

```
public void deleteTest(){
      ContentResolver resolver = getContext().getContentResolver();
      Uri uri =
Uri.parse("content://com.itheima.person/preson/delete");
      int count = resolver.delete(uri, "id=?", new String[]{"1"});
      System.out.println("删除的个数为:  "+count);
   }
```

测试 query（单个）方法

```
public void queryOne(){
      ContentResolver resolver = getContext().getContentResolver();
      Uri uri = Uri.parse("content://com.itheima.person/person/2");
      Cursor cursor = resolver.query(uri, new
String[]{"name","age","phone","address"}, null, null, null);
      while(cursor.moveToNext()){
         String name = cursor.getString(0);
         int age = cursor.getInt(1);
         String phone = cursor.getString(2);
         String address = cursor.getString(3);
         System.out.println(name+"/"+age+"/"+phone+"/"+address);
      }
      cursor.close();
   }
```

测试 query（多个）方法

```
public void queryAll(){
      ContentResolver resolver = getContext().getContentResolver();
      Uri uri = Uri.parse("content://com.itheima.person/person");
      Cursor cursor = resolver.query(uri, new
String[]{"name","age","phone","address"}, null, null, null);
      while(cursor.moveToNext()){
          String name = cursor.getString(0);
          int age = cursor.getInt(1);
          String phone = cursor.getString(2);
          String address = cursor.getString(3);
          System.out.println(name+"/"+age+"/"+phone+"/"+address);
      }
      cursor.close();
   }
```

2.案例-短信的备份和恢复（★★★★）

Android 系统中提供了一系列的内容提供者，通过调用他们，可以获取一些系统的信息，比如:短信信息，联系人信息等。

需求：在应用界面有备份短信、恢复短信两个按钮，点击该按钮会将当前短信备份到 XML 文件中，同时 XML 文件被存储在存储卡中。点击恢复短信按钮，程序会将 XML 文件中的短信插入到手机中存储短信的数据库中。在备份和恢复的过程中有进度条显示完成情况。

2.1 准备知识

打开 Android 源码，查看 packages\providers\路径下的工程，这些就是 Android 系统中的内容提供者，其中 TelephonyProvider 就是短信的内容提供者文件。

- 打开 TelephonyProvider 下的 src 文件 ,查看 java 文件 ,其中的 SmsProvider.java 即短信息内容提供者逻辑代码。UriMatcher 一般在静态代码块中进行初始化操作，查找静

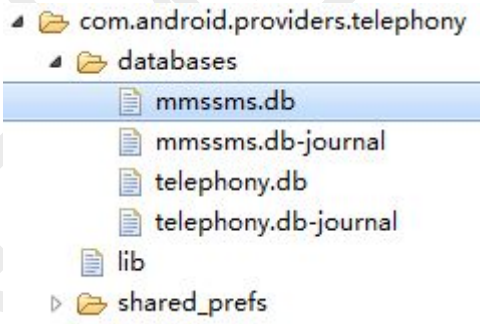
态代码块，找到的逻辑代码如下：

```
private static final UriMatcher sURLMatcher =
                new UriMatcher(UriMatcher.NO_MATCH);

        static {
            sURLMatcher.addURI("sms", null, SMS_ALL);
            sURLMatcher.addURI("sms", "#", SMS_ALL_ID);
            sURLMatcher.addURI("sms", "inbox", SMS_INBOX);
            sURLMatcher.addURI("sms", "inbox/#", SMS_INBOX_ID);
            sURLMatcher.addURI("sms", "sent", SMS_SENT);
            sURLMatcher.addURI("sms", "sent/#", SMS_SENT_ID);
            sURLMatcher.addURI("sms", "draft", SMS_DRAFT);
            sURLMatcher.addURI("sms", "draft/#", SMS_DRAFT_ID);
            sURLMatcher.addURI("sms", "outbox", SMS_OUTBOX);
            sURLMatcher.addURI("sms", "outbox/#", SMS_OUTBOX_ID);
            sURLMatcher.addURI("sms", "undelivered",
SMS_UNDELIVERED);
            sURLMatcher.addURI("sms", "failed", SMS_FAILED);
            sURLMatcher.addURI("sms", "failed/#", SMS_FAILED_ID);
            sURLMatcher.addURI("sms", "queued", SMS_QUEUED);
            sURLMatcher.addURI("sms", "conversations",
SMS_CONVERSATIONS);
            sURLMatcher.addURI("sms", "conversations/*",
SMS_CONVERSATIONS_ID);
            sURLMatcher.addURI("sms", "raw", SMS_RAW_MESSAGE);
            sURLMatcher.addURI("sms", "attachments", SMS_ATTACHMENT);
            sURLMatcher.addURI("sms", "attachments/#",
SMS_ATTACHMENT_ID);
            sURLMatcher.addURI("sms", "threadID", SMS_NEW_THREAD_ID);
            sURLMatcher.addURI("sms", "threadID/*",
SMS_QUERY_THREAD_ID);
            sURLMatcher.addURI("sms", "status/#", SMS_STATUS_ID);
            sURLMatcher.addURI("sms", "sr_pending",
SMS_STATUS_PENDING);
            sURLMatcher.addURI("sms", "icc", SMS_ALL_ICC);
            sURLMatcher.addURI("sms", "icc/#", SMS_ICC);
            //we keep these for not breaking old applications
            sURLMatcher.addURI("sms", "sim", SMS_ALL_ICC);
            sURLMatcher.addURI("sms", "sim/#", SMS_ICC);
```

```
sConversationProjectionMap.put(Sms.Conversations.SNIPPET,
                "sms.body AS snippet");

sConversationProjectionMap.put(Sms.Conversations.THREAD_ID,
                "sms.thread_id AS thread_id");

sConversationProjectionMap.put(Sms.Conversations.MESSAGE_COUNT,
                "groups.msg_count AS msg_count");
            sConversationProjectionMap.put("delta", null);
        }
    }
```

在数据库中 sms 表就是用于存储短信的，所以通过查找系统源码，可以确定短信息内容提供者的 Uri 应该为:”content://sms”

查看 Android 模拟器下的 com.android.providers.telephony ，查看其 mmssms.db 文件



打开数据库，其中 sms 表存储的就是短信的数据，其存储格式如下：

RecNo	_id	thread_id	address	person	date	date_sent	protocol	read	status	type	reply_path_present	subject	body	servi
Click here to define a filter														
1	1	1	5556	<null>	1410697956692	1410697959000	0	1	-1	1	0	<null>	hello	<null
2	2	1	5556	<null>	1410697964854	1410697967000	0	1	-1	1	0	<null>	calling	<null
3	3	2	12343	<null>	1410697979010	1410697981000	0	1	-1	1	0	<null>	hello world	<null
4	4	1	5556	<null>	1410697998401	0	<null>	1	-1	2	<null>	<null>	ok	<null

其中，address 存储的是联系人号码，date 是发送日期，type 对应短信的类型（发送/接收）,body 是短信的主体内容，准备备份这四项。

2.2 实现步骤

1 新创建一个新的 Android 工程《短信备份 And 恢复》，包名：com.itheima.smsbackup。在 com.itheima.smsbackup.bean 包中创建 Sms 类，用于封装短信数据。

```
public class Sms {
    private String body;
    private String address;
    private String date;
    private String type;
//省略 setter&getter
}
```

2 创建 XML 工具类，用于生成 XML 和解析 XML。

```
public class Sms {
    private String body;
    private String address;
    private String date;
    private String type;
//省略 setter&getter
}
```

3 编写界面布局文件，这里使用默认生成的布局文件 activity_main.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <TextView
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="短信备份和恢复" />
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="备份短信"
        android:onClick="backupSms"
        android:id="@+id/bt_backup"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="恢复短信"
        android:onClick="reverseSms"
        android:id="@+id/bt_reverse"
        />
    <ProgressBar
        android:id="@+id/pb"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="?android:attr/progressBarStyleHorizontal"
        />

</LinearLayout>
```

4 编写默认的 MainActivity 类，实现核心方法。

备份短信功能代码清单

```
public void backupSms(View view) {
      //如果当前正在备份，则返回。
      if (backupFlag) {
          Toast.makeText(this, "当前正在备份中。。。请稍后再操作",
0).show();
          return;
      }
      //开辟一个新的线程来处理业务，因为短信备份是耗时操作，如果不放在子
线程中操作可能会导致 ANR 异常
      new Thread(new Runnable() {

          @Override
          public void run() {
```

```
//将标记位设置为 true
                backupFlag = true;
                //子线程中创建一个 Looper 对象
                Looper.prepare();
                //获取 ContentResolver 对象
                ContentResolver resolver = getContentResolver();
                //编辑短信访问的 uri
                Uri uri = Uri.parse("content://sms");
                //执行查询语句
                Cursor cursor = resolver.query(uri, new String[]
{ "address", "date", "body", "type" }, null, null, null);
                List<Sms> list = new ArrayList<Sms>();
                //获取短信的总共条数
                int count = cursor.getCount();
                //设置进度条的最大值
                pb.setMax(count);
                int i = 0;
                while (cursor.moveToNext()) {
                    Sms sms = new Sms();
                    String address = cursor.getString(0);
                    String date = cursor.getString(1);
                    String body = cursor.getString(2);
                    String type = cursor.getString(3);
                    sms.setAddress(address);
                    sms.setBody(body);
                    sms.setDate(date);
                    sms.setType(type);
                    list.add(sms);
                    //更新进度条
                    pb.setProgress(++i);
                    //这里是为了方便演示备份的效果加上线程等待
                    SystemClock.sleep(50);
                }
                try {
                    //通过自定义的 Sms2XmlUtil 工具类将短信保存到 xml 中
                    Sms2XmlUtil.sms2Xml(list);
                } catch (Exception e) {
                    e.printStackTrace();
                    Toast.makeText(MainActivity.this, "备份短信失败。" +
e.getLocalizedMessage(), 1).show();
                }
```

```
//关闭游标
                cursor.close();
                Toast.makeText(MainActivity.this,"短信备份成功！本次备份了" + i + "条短信。", 1).show();
                //循环消息
                Looper.loop();
                //将标记设置为flase
                backupFlag = false;
            }
        }).start();
}
```

恢复短信功能代码清单

```
public void reverseSms(View view) {
        if (reverseFlag) {
            Toast.makeText(this, "当前短信正在备份中，请稍后再操作。", 0).show();
            return;
        }
        new Thread(new Runnable() {
            @Override
            public void run() {
                Looper.prepare();
                ContentResolver resolver = getContentResolver();
                Uri url = Uri.parse("content://sms");
                List<Sms> list = null;
                try {
                    list = Sms2XmlUtil.xml2Sms();
                    pb.setMax(list.size());
                    pb.setProgress(0);
                } catch (Exception e) {
                    e.printStackTrace();
                    Toast.makeText(MainActivity.this, "短信恢复失败！" + e.getLocalizedMessage(), 1).show();
                    return;
                }
                int i = 0;
                for (Sms s : list) {
                    ContentValues values = new ContentValues();
                    values.put(Sms2XmlUtil.SMS_TAG_ADDRESS,
```

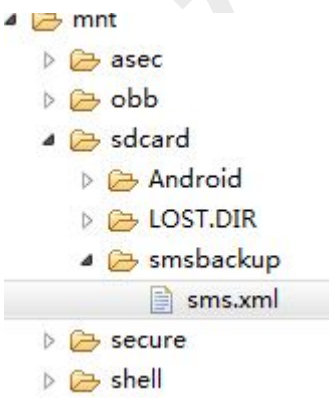
```
s.getAddress());
                values.put(Sms2XmlUtil.SMS_TAG_BODY, s.getBody());
                values.put(Sms2XmlUtil.SMS_TAG_DATE, s.getDate());
                values.put(Sms2XmlUtil.SMS_TAG_TYPE, s.getType());
                Uri uri = resolver.insert(url, values);
                System.out.println("插入的 id: " +
ContentUris.parseId(uri));
                pb.setProgress(++i);
                SystemClock.sleep(50);
            }
            Toast.makeText(MainActivity.this,"短信恢复成功！本次恢复
了" + i + "条短信。", 1).show();
            reverseFlag = false;
            Looper.loop();
        }
      }).start();

   }
```

5 为当前工程添加权限。在 AndroidManifest.xml 中添加权限声明。

```
 <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_SMS"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
```

6 将本工程部署到模拟器上，运行，并测试。发现我们成功将短信备份到了存储卡中。

通过 DDMS，我们可以看到在 mnt/sdcard/smsbackup 目录下多了 sms.xml 文件。



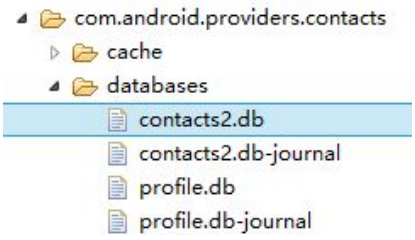
程序运行界面比较简单，效果如下图：

Tips：在该工程中，短信的恢复逻辑比较简单，只是把 sms.xml 文件中的所有短信插入到短信数据库中，其实如果短信数据库中还有相同的短信的时候我们这样处理的结果是数据库中有了 2 份相同的短信，因此比较好的做法是在插入数据库之前先判断该短信是否存在，如果存在则不插入。

3.操作系统联系人（★★★★）

3.1 准备知识

通过 DDMS，查看 Android 模拟器下的 com.android.providers.contacts 包下的数据库，查看其 contact2.db 数据库的内容。



查看数据库，其中 raw_contacts 表存放的是联系人条数信息，data 表中存放的是 raw_contacts 中的每一条 id 对应的具体信息，不同类型的信息由 mimetype_id 来标识。

raw_contacts 表：

Rec...	_id	acc...	sour...	ra...	ver...	dirty	dele...	contact_id	aggr...	ag...	custom...	send...	ti...	last_ti...	st...	displ...	displa...	dis...	phon...	ph...	sort_key	sort_ke
Click here to define a filter																						
1	1	1	<null>	0	2	1	0	1	0	0	<null>	0	0	<null>	0	张三	张三	40	<null>	0	ZHANG 张 SAN 三	ZHANG
2	2	1	<null>	0	2	1	0	2	0	0	<null>	0	0	<null>	0	李四	李四	40	<null>	0	LI 李 SI 四	LI 李 SI

data 表：

Rec...	_id	package_id	mimetype_id	raw_contact_id	is_r...	is_pr...	is_sup...	data_v...	data1	data2	data3	data4	data5	data6
Click here to define a filter														
1	1	<null>	5	1	0	0	0	0	1 234-456-7823	2	<null>	+12344567823	<null>	<null>
2	2	<null>	8	1	0	0	0	0	北京	1	<null>	北京	<null>	<null>
3	3	<null>	1	1	0	0	0	0	zhangsan@163.com	1	<null>	<null>	<null>	<null>
4	4	<null>	7	1	0	0	0	0	张三	三	张	<null>	<null>	<null>
5	5	<null>	5	2	0	0	0	0	1 123-231-1442	2	<null>	<null>	<null>	<null>
6	6	<null>	8	2	0	0	0	0	nanjing	1	<null>	nanjing	<null>	<null>
7	7	<null>	1	2	0	0	0	0	lisi@163.com	1	<null>	<null>	<null>	<null>
8	8	<null>	7	2	0	0	0	0	李四	四	李	<null>	<null>	<null>

mimetypes 表：

RecNo	_id	mimetype
Click here to define a filter		
1	1	vnd.android.cursor.item/email_v2
2	2	vnd.android.cursor.item/im
3	3	vnd.android.cursor.item/nickname
4	4	vnd.android.cursor.item/organization
5	5	vnd.android.cursor.item/phone_v2
6	6	vnd.android.cursor.item/sip_address
7	7	vnd.android.cursor.item/name
8	8	vnd.android.cursor.item/postal-address_v2
9	9	vnd.android.cursor.item/identity
10	10	vnd.android.cursor.item/photo
11	11	vnd.android.cursor.item/group_membership

打开 Android 源码，查看 packages\providers\ 路径下的文件，其中

ContactsProvider 就是联系人的内容提供者。

1．打开清单文件，寻找联系人的内容提供者对应的是哪个 java 文件。

```
<provider android:name="ContactsProvider2"
          android:authorities="contacts;com.android.contacts"
          android:label="@string/provider_label"
          android:multiprocess="false"

android:readPermission="android.permission.READ_CONTACTS"

android:writePermission="android.permission.WRITE_CONTACTS">
          <path-permission
                 android:pathPrefix="/search_suggest_query"

android:readPermission="android.permission.GLOBAL_SEARCH" />
          <path-permission
                 android:pathPrefix="/search_suggest_shortcut"

android:readPermission="android.permission.GLOBAL_SEARCH" />
          <path-permission
                 android:pathPattern="/contacts/.*/photo"

android:readPermission="android.permission.GLOBAL_SEARCH" />
          <grant-uri-permission android:pathPattern=".*" />
       </provider>
```

2．打开 ContactsProvider2.java 文件，查看此内容提供者的 uri 路径信息

```
        matcher.addURI(ContactsContract.AUTHORITY, "contacts",
      CONTACTS);
        matcher.addURI(ContactsContract.AUTHORITY, "contacts/#",
      CONTACTS_ID);
        matcher.addURI(ContactsContract.AUTHORITY,
      "contacts/#/data", CONTACTS_DATA);
        matcher.addURI(ContactsContract.AUTHORITY,
      "raw_contacts", RAW_CONTACTS);
         matcher.addURI(ContactsContract.AUTHORITY,
"raw_contacts/#", RAW_CONTACTS_ID);
         matcher.addURI(ContactsContract.AUTHORITY,
"raw_contacts/#/data", RAW_CONTACTS_DATA);
```

3．根据源码，确定内容提供者的 Uri 信息为：

操作 raw_contacts 表的 Uri:

```
content://com.android.contacts/raw_contacts
```

操作 data 表的 Uri：

```
content://com.android.contacts/data
```

4．操作数据库表时 注意：由于 contacts2.db 数据库使用了视图，所以操作数据库表时，表结构有所改变，注意操作时要操纵的列的列名已经改变。

比如：data 表在查询的时候没有 mimetype_id,取代的是 mimetype

3.2 操作系统联系人

步骤：

1. 操作 raw_contacts 表，获取全部的 id
2. 根据获取到的每一条 id 去查询 data 表中的数据
3. 将查询到的展示是用户界面
4. 通过代码给系统联系人插入一条联系人信息

1 新建一个 Android 工程《操作系统联系人》。包名：com.itheima.contacts。

2 使用默认生成的布局文件和 MainActivity 类。在布局文件里将用户联系人展示在 ListView 中，因此我们需要为 ListView 的 Item 创建一个布局文件，由于该布局文件比较简单，因此我们只给出效果图，布局文件清单就不再给出。activity_main.xml 文件清单：

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <ListView
        android:id="@+id/lv"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" >
    </ListView>

    <LinearLayout
        android:layout_gravity="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <Button
            android:layout_width="0dp"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:onClick="queryContacts"
            android:text="获取联系人" />

        <Button
            android:layout_width="0dp"
            android:layout_weight="1"
            android:layout_height="wrap_content"
            android:onClick="insertContacts"
            android:text="插入联系人" />
    </LinearLayout>

</LinearLayout>
```

想想还是将 list_item.xml 的布局给出吧，哎，任性了。

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:baselineAligned="false"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
    android:orientation="horizontal"
    >
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:layout_weight="1"
        >
        <TextView
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="联系人"
            android:textColor="#0000ff"
            />
        <TextView
            android:id="@+id/tv_name"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="张三"
            android:textColor="#0000ff"
            />
    </LinearLayout>
    <LinearLayout
        android:layout_height="match_parent"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:orientation="vertical"
        >
         <TextView
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="电话"
            android:textColor="#ff0000"
            />
        <TextView
            android:id="@+id/tv_phone"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="12333333"
            android:textColor="#ff0000"
            />
```

```
</LinearLayout>
    <LinearLayout
        android:layout_height="match_parent"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:orientation="vertical"
        >
         <TextView
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="邮件"
            android:textColor="#00ff00"
            />
        <TextView
            android:id="@+id/tv_email"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="12333333@qq.com"
            android:textColor="#00ff00"
            />

    </LinearLayout>
</LinearLayout>
```

3 编写 MainActivity 类代码，在这里我们用到了自定义的 Contact 对象，用于封装属性，比较简单因此就不再给出 Contact 类代码清单。

```
public class MainActivity extends Activity {

    private ListView lv;
    private List<Contact> list;
    private MyAdapter adapter;
    private Handler handler = new Handler() {
        public void handleMessage(android.os.Message msg) {
            Toast.makeText(MainActivity.this, "数据获取成功。",
1).show();
            //更新数据
            adapter.notifyDataSetChanged();
        };
```

```
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lv = (ListView) findViewById(R.id.lv);
        list = new ArrayList<Contact>();
        adapter = new MyAdapter();
        lv.setAdapter(adapter);
    }
    /**
     * 查询所有的系统联系人，这里的业务放在子线程中操作
     */
    public void queryContacts(View view) {
        new Thread(new Runnable() {

            @Override
            public void run() {
                ContentResolver contentResolver = getContentResolver();
                Uri uri =
Uri.parse("content://com.android.contacts/raw_contacts");
                Cursor cursor = contentResolver.query(uri, new String[]
{ "contact_id" }, null, null, null);
                list.clear();
                int i = 0;
                while (cursor.moveToNext()) {
                    i++;
                    //先从 raw_contacts 表中获取 contact_id
                    String contact_id = cursor.getString(0);
                    if (TextUtils.isEmpty(contact_id)) {
                        continue;
                    }
                    //根据 contact_id 从 data 表中查询具体的数据
                    Cursor cursor2 =
contentResolver.query(Uri.parse("content://com.android.contacts/da
ta"), new String[] { "data1", "mimetype" }, "contact_id=?", new
String[] { contact_id }, null);
                    Contact contact = new Contact();
                    while (cursor2.moveToNext()) {
                        String data = cursor2.getString(0);
```

```
                    String mimetype = cursor2.getString(1);
                    if
("vnd.android.cursor.item/name".equals(mimetype)) {
                        contact.setName(data);
                    } else if
("vnd.android.cursor.item/phone_v2".equals(mimetype)) {
                        contact.setPhone(data);
                    } else if
("vnd.android.cursor.item/email_v2".equals(mimetype)) {
                        contact.setEmail(data);
                    } else {
                        contact.setOther(data);
                    }
                }
                list.add(contact);
                cursor2.close();
            }
            cursor.close();
            //发送一个空消息，更新 ListView
            handler.sendEmptyMessage(RESULT_OK);
        }
    }).start();

}

class MyAdapter extends BaseAdapter {

    @Override
    public int getCount() {
        return list.size();
    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return position;
    }
```

```
        /**
         * 这里面通过 ViewHolder 类将其他子属性值绑定在 View 上面，这里对
ListView 作了进一步的优化处理
         */
        @Override
        public View getView(int position, View convertView, ViewGroup
parent) {
            View view;
            ViewHolder holder;
            if (convertView != null) {
                view = convertView;
                holder = (ViewHolder) view.getTag();
            } else {
                view = View.inflate(MainActivity.this,
R.layout.list_item, null);
                holder = new ViewHolder();
                holder.tv_name = (TextView)
view.findViewById(R.id.tv_name);
                holder.tv_phone = (TextView)
view.findViewById(R.id.tv_phone);
                holder.tv_email = (TextView)
view.findViewById(R.id.tv_email);
                view.setTag(holder);
            }
            Contact contact = list.get(position);
            holder.tv_name.setText(contact.getName());
            holder.tv_email.setText(contact.getEmail());
            holder.tv_phone.setText(contact.getPhone());
            return view;
        }
    }

    class ViewHolder {
        TextView tv_name;
        TextView tv_email;
        TextView tv_phone;
    }
    /**
     * 往系统联系人表中插入一条数据，这里为了方便演示，我们直接插入一条固定
的数据
     */
```

```
    public void insertContacts(View view) {
        //创建一个自定义的 Contact 类，将要网系统联系人表中插入的字段封装起
来
        Contact contact = new Contact();
        contact.setEmail("itheima@itcast.cn");
        contact.setName("王二麻子"+new Random().nextInt(1000));
        contact.setPhone("9999999"+new Random().nextInt(100));
        contact.setOther("北京市中关村软件园");
        //获取 ContentResolver 对象
        ContentResolver resolver = getContentResolver();
        //操作 raw_contacts 表的 uri
        Uri raw_uri =
Uri.parse("content://com.android.contacts/raw_contacts");
        //操作 data 表的 uri
        Uri data_uri =
Uri.parse("content://com.android.contacts/data");
        //在插入数据之前先查询出当前最大的 id
        Cursor cursor = resolver.query(raw_uri, new String[]
{ "contact_id" }, null, null, "contact_id desc limit 1");
        int id = 1;
        if (cursor!=null) {
            boolean moveToFirst = cursor.moveToFirst();
            if (moveToFirst) {
                id = cursor.getInt(0);
            }
        }
        cursor.close();
        //要插入数据的 contact_id 值
        int newId = id + 1;
        // 给 raw_contact 表中添加一天记录
        ContentValues values = new ContentValues();
        values.put("contact_id", newId);
        resolver.insert(raw_uri, values);
        // 在 data 表中添加数据
        // 添加 name
        values = new ContentValues();
        values.put("raw_contact_id", newId);
        values.put("mimetype", "vnd.android.cursor.item/name");
        values.put("data1", contact.getName());
        resolver.insert(data_uri, values);
        // 添加 phone
        values = new ContentValues();
```

```
        values.put("raw_contact_id", newId);
        values.put("mimetype", "vnd.android.cursor.item/phone_v2");
        values.put("data1", contact.getPhone());
        resolver.insert(data_uri, values);
        // 添加地址
        values = new ContentValues();
        values.put("raw_contact_id", newId);
        values.put("mimetype",
"vnd.android.cursor.item/postal-address_v2");
        values.put("data1", contact.getOther());
        resolver.insert(data_uri, values);
        // 添加 email
        values = new ContentValues();
        values.put("raw_contact_id", newId);
        values.put("mimetype", "vnd.android.cursor.item/email_v2");
        values.put("data1", contact.getEmail());
        resolver.insert(data_uri, values);

        Toast.makeText(this, "成功插入联系人" + contact, 0).show();

    }

}
```

4 在 AndroidManifest.xml 中添加添加如下权限：

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission
android:name="android.permission.WRITE_CONTACTS"/>
```

5 部署该工程于模拟器。点击获取联系人，然后在点击插入联系人，然后在点击获取联系人。发现我们不仅成功的将所有的联系人读取出来了同时也将新联系人插入到了数据库中。

运行效果图如下：

4.ContentObserver 简介（★★★）

ContentObserver——内容观察者，目的是观察(捕捉)特定 Uri 引起的数据库的变化，继而做一些相应的处理，它类似于数据库技术中的触发器(Trigger)，当 ContentObserver 所观察的 Uri 发生变化时，便会触发它。触发器分为表触发器、行触发器，相应地 ContentObserver 也分为“表“ContentObserver、“行”ContentObserver，当然这是与它所监听的 Uri MIME Type 有关的。

学习过 Content Provider(内容提供者)的我们都知道，可以通过 UriMatcher 类注册不

同类型的 Uri，可以通过这些不同的 Uri 来查询不同的结果。根据 Uri 返回的结果，Uri Type 可以分为：返回多条数据的 Uri、返回单条数据的 Uri。

使用 ContentObserver 的步骤我这里总结如下：

1. 首先创建一个 ContentObserver 的子类，然后实现里面的 onChange 方法，监听的 Uri 中的数据发生变化的时候，会调用 onChange 方法。
2. 注册 ContentObserver。

在下面第 5 章节中，我将用一个案例来演示 ContentObserver 的用法。

5.案例-短信窃听（★★★★）

用户使用系统自带的短信程序发送短信，程序会通过 ContentProvider 把短信保存进数据库，并且发出一个数据变化通知，使用 ContentObserver 对数据变化进行监听，在用户发送短信时，就会被 ContentObserver 窃听到短信。

发出的短信有这样几个过程：草稿箱->发件箱->已发送。所以只需查询发件箱中的信息即可，处于正在发送状态的短信放在发送箱中。

需求：创建一个应用程序，监听用户发送的短信。

1 新创建一个 Android 工程《短信窃听》，包名：com.itheima.contentObserver。

2 新创建一个 SmsContentObserver 类继承 ContentObserver 类，覆写 onChange 方法。在该方法中实现核心业务逻辑。

```
public class SmsContentObserver extends ContentObserver {
    //声明一个 Content 对象，在构造函数中对其进行实例化
    private Context context;
    private Handler handler;
    //声明构造函数
```

```
    public SmsContentObserver(Context context,Handler handler){
        super(handler);
        this.context = context;
        this.handler = handler;
    }
    //覆写父类 onChange 方法
    @Override
    public void onChange(boolean selfChange) {
        //通过查看发件箱中的短信即可观察到新短信的发送
        Uri uri = Uri.parse("content://sms/outbox");
        //获取 Context 对象的 ContentResolver 对象
        ContentResolver resolver = context.getContentResolver();
        String[] projection = {"address","body","date"};
        //获取发件箱中的短信
        Cursor cursor = resolver.query(uri, projection, null, null,
null);
        if(cursor!=null&&cursor.moveToNext()){
            String address = cursor.getString(0);
            String body = cursor.getString(1);
            Long date = cursor.getLong(2);
            //调用 DateFormat 的方法将 long 类型的时间转化为系统相关时间
            String dateStr =
DateFormat.getTimeFormat(context).format(date);
            //打印观察到正在发送的短信
            System.out.println("address="+address+" body="+body+"
date="+dateStr);
        }
        cursor.close();
    }
}
```

3 在 MainActivity 类中注册 SmsContentObserver。

```
public class MainActivity extends Activity {
    private SmsContentObserver observer;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //实例化一个自定义的 SMSContentObserver 对象
```

```
        observer = new SmsContentObserver(this, null);
        //操作系统短信的 uri
        Uri uri = Uri.parse("content://sms");
        /**
         * 获取 ContentResolver,然后注册 ContentObserver
         */
        getContentResolver().registerContentObserver(uri, true,
observer);
    }

}
```

4 因为我们读取系统短信了，因此需要在 AndroidManifest.xml 中添加权限。

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

5 将该工程部署在模拟器上并运行，然后打开短息功能，给 5557 模拟器（可以随意指定个号码，不需要启动 5557 模拟器）发送一条短信。观察 LogCat 发现，我们的 ContentObserver 成功监听到了短信内容。

```
address=555-7 body=hello 5557 i am 5554! date=下午 4:47
```

6.使用 ContentObserver 监听自定义的 ContentProvider（★★★）

在第 5 章节中，我们通过 ContentObserver 实现了对系统发送短信的监听。这是因为系统短信数据数据库中插入新的短信的时候通过 ContentResolver 调用了 notifyChange 方法，正是该方法对外发出了消息，这样我们的 ContentObserver 才监听到了短信的发送。

那么我们也可以自定义我们的 ContentProvider，在 ContentProvider 中发送消息，被我们的 ContentObserver 监听到。

为了方便演示，我们直接使用该文档 1.1 章节中创建的工程《MyContentProvider》。

我们修改 PersonContentProvider 类中的 insert 方法，当有新用户插入数据库的时候发出消息以被 ContentObserver 监听。

```
public Uri insert(Uri uri, ContentValues values) {
        SQLiteDatabase db = openHelper.getWritableDatabase();
        //新插入对象的 id
        long id = db.insert(TABLE_PERSON_NAME, null, values);
        db.close();
        //声明一个 uri，通过这个 uri ContentObserver 可以进行监听
        Uri notifyUri = Uri.parse("com.itheima.person");
        /*
         * notifyChange 方法中第一个参数指定一个被监听的 uri
         * 第二个参数是指定 ContentObserver，如果不为 null 则代表只有指定的
ContentObserver 可以接收到消息
         * 为 null 则所有的 ContentObserver 都有机会接收到该消息
         */
        getContext().getContentResolver().notifyChange(notifyUri,
null);
        //使用 ContentUris 工具类将 id 追加到 uri 中，返回给客户
        return ContentUris.withAppendedId(uri, id);
    }
```

同样的，接下来同监听短信一样，我们已经可以通过自定义 ContentObserver 对我们的 person 数据库的添加动作进行监听了。由于监听的原理跟第 5 章完成一样，因此这里就不再给出详细的代码。

至此，本文档完！

2014 年 12 月 17 日星期三 1:31:44
北京市海淀区东北旺中路东馨园