Android移动开发

Android Mobile Application Development

第13讲 Android组件

吴以凡

计算机学院 一教505

yfwu@hdu.edu.cn

Android组件

Android组件

- 组件是可以调用的基本功能模块，Android应用程序就是由一个或多个基本组件组成的
- Android系统有四个重要的组件，分别是
  - 活动（Activity）
  - 服务（Service）
  - 广播（Broadcast）
  - 内容提供器（ContentProvider）

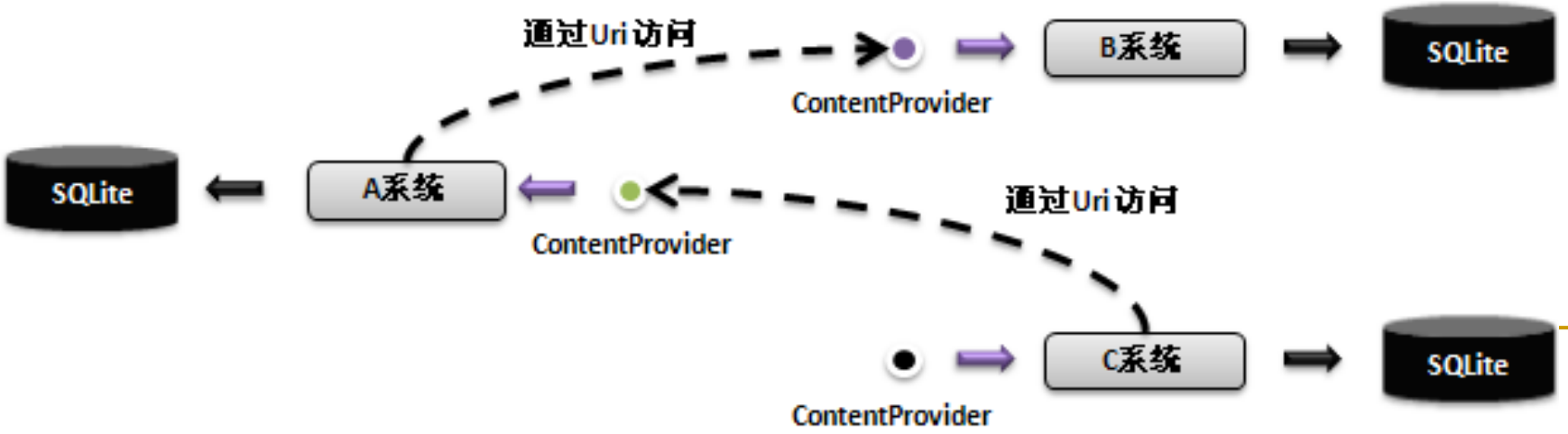
ContentProvider

ContentProvider简介

- Android为我们提供了ContentProvider来实现数据的共享
  - 程序如果想让别的程序可以操作自身的数据，就定义自己的 ContentProvider
  - 然后在AndroidManifest.XML中注册
  - 其他application可以通过获取ContentResolver来操作这个程序的数据

ContentProvider简介

- 每个应用程序的数据都是采用私有的形式进行操作
  - 无论数据是文件保存还是数据库保存，都不能被外部应用程序所访问
- ContentProvider类是不同应用程序之间进行数据交换的接口
  - 将不同的应用程序的数据操作标准统一起来，并且将各个应用程序的数据操作标准暴露给其他应用程序



ContentProvider简介

- Android自身也提供了几个现成的Content provider：
  - Contacts
  - Browser
  - CallLog
  - Settings
  - MediaStore
- 可以在Android.provider包下面找到Android提供的ContentProvider

ContentProvider的创建

- 开发ContentProvider需要两步
  - 首先创建一个它的子类，该类需要实现它的抽象方法，如query()、insert()、update()和delete()等方法
  - 然后在AndroidManifest.xml文件中注册ContentProvider

ContentProvider的创建

- 第一步：创建一个继承ContentProvider的类，并实现其抽象方法

```
public class TestContentProvider extends ContentProvider {
    public boolean onCreate() {
        return false;
    }
    public Cursor query( Uri uri, String[] strings, String s, String[] strings1, String s1) {
        return null;
    }
    public String getType(Uri uri) {
        return null;
    }
    public Uri insert(Uri uri, ContentValues contentValues) {
        return null;
    }
    public int delete(Uri uri, String s, String[] strings) {
        return 0;
    }
    public int update(Uri uri, ContentValues contentValues, String s, String[] strings) {
        return 0;
    }
}
```

ContentProvider的创建

- ContentProvider类的抽象类及常用的操作方法

方法名称	描述
public abstract boolean onCreate()	当启动此组件的时候调用
public abstract int delete(Uri uri, String selection, String[] selectionArgs)	根据指定的Uri删除数据，并返回删除数据的行数
public abstract String getType(Uri uri)	根据指定的Uri，返回操作的MIME类型
public abstract Uri insert(Uri uri, ContentValues values)	根据指定的Uri进行增加数据的操作，并且返回增加后的Uri，在此Uri中会附带有新数据的_id
public abstract Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)	根据指定的Uri执行查询操作，所有的查询结果通过Cursor对象返回
public abstract int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)	根据指定的Uri进行数据的更新操作，并返回更新数据的行数
public final Context getContext()	返回Context对象

ContentProvider的创建

- 第二步：在AndroidManifest.xml中注册ContentProvider
  - android.authorities为访问本provider的路径
  - android:name为ContentProvider子类的类名

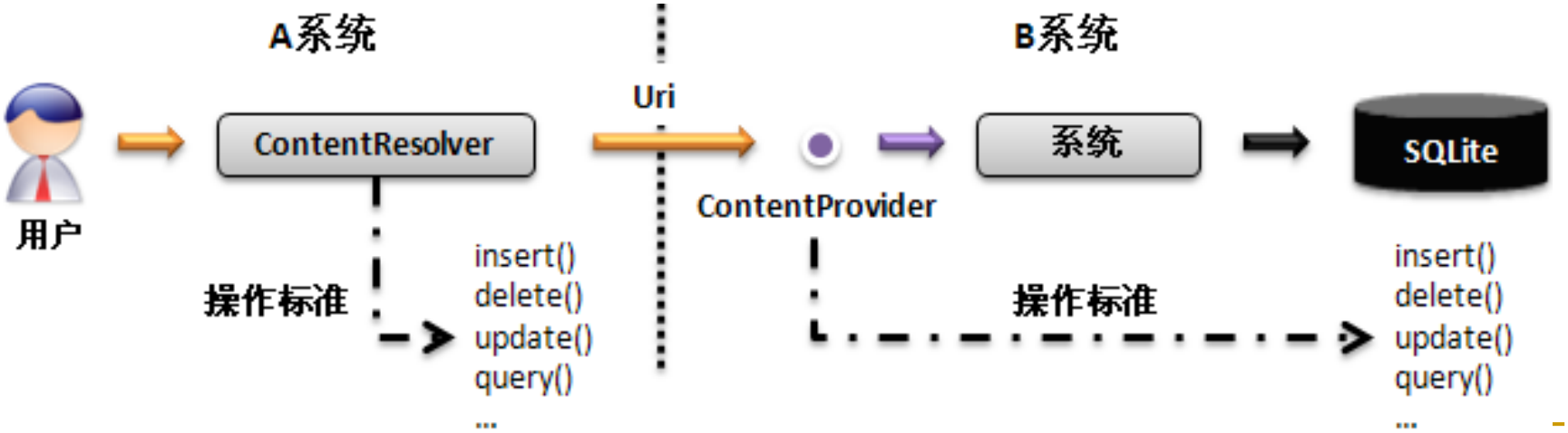
```
<provider
android:authorities="cn.hdu.contentproviderexample.TestContentProvider "
android:name=".TestContentProvider">
</provider>
```

Uri简介

- 参数Uri uri代表了数据的操作方法。URI形式：content://authority/path/id
  - content://是标准前缀，表明了这个数据被内容提供者管理
  - authority：是在清单文件注册的android:authorities属性值，该值唯一，表明了当前的内容提供者，使用完全包名
  - path：提供者内部的虚拟目录，用于标识被请求的数据类型
  - id 被请求的特定记录的主键。省略它表示所有记录
- Uri示例:
  - content://media/internal/images：返回设备上存储的所有图片
  - content://contacts/people/：返回设备上的所有联系人信息
  - content://contacts/people/45：返回单个结果（联系人信息中ID为45的联系人记录）

ContentResolver简介

- 应用程序通过ContentProvider暴露自己的数据
- 其他应用程序使用ContentResolver类对暴露的数据进行操作
- ContentResolver类的操作方法与ContentProvider是一一对应的，当用户调用了ContentResolver类的方法时实际上就相当于调用了ContentProvider类中的对应方法



ContentResolver简介

- ContentResolver类的常用操作方法

方法名称	描述
public final int delete(Uri url, String where, String[] selectionArgs)	调用指定ContentProvider对象中的delete()方法
public final String getType(Uri url)	调用指定ContentProvider对象中的getType()方法
public final Uri insert(Uri url, ContentValues values)	调用指定ContentProvider对象中的insert()方法
public final Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)	调用指定ContentProvider对象中的query()方法
public final int update(Uri uri, ContentValues values, String where, String[] selectionArgs)	调用指定ContentProvider对象中的update()方法

ContentResolver简介

- Activity类对ContentResolver类的操作方法
  - 通过android.app.Activity类的getContentResolver()方法可以取得ContentResolver类的实例化对象
- 由于使用ContentProvider暴露数据时，提供了相应操作的Uri，所以在使用ContentResolver获取数据的时候，需要指定相应的的Uri

```
//得到ContentResolver对象
ContentResolver cr = getContentResolver();
//取得电话本中开始一项的光标
//ContactsContract.Contacts.CONTENT_URI是手机通讯录的Uri
Cursor cursor=cr.query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
```

实例：获取手机通讯录

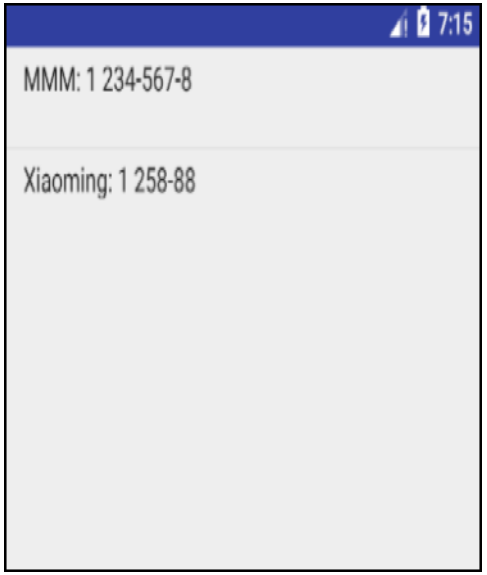
- Android的API中提供了一个Contacts类用于管理联系人，而且还为联系人管理提供了ContentProvider, 这就允许其他程序以ContentResolver来管理联系人数据。

- Android 对联系人管理ContentProvider的几个Uri如下：
  - ContactsContract.Contacts.CONTENT_URI： 管理联系人的Uri
  - ContactsContract.CommonDataKinds.Phone.CONTENT_URI： 管理联系人的电话的Uri

实例：获取手机通讯录

- (1) activity_main.xml文件
  - 采用线性布局的方式，添加ListView用来显示获取到的姓名和手机号

```
<ListView
   android:id="@+id/show_people"
   android:layout_width="match_parent"
   android:layout_height="wrap_content">
</ListView>
```



- (2) MainActivity中的代码
  - 修改MainActivity的代码，定义获取系统通讯录的Uri，然后获取电话本开始一项的Uri，最后逐行读取，把信息存储到List数组中，最终显示到ListView

实例：获取手机通讯录

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //定义List的对象用于保存数据
    List<String> string;
    setContentView(R.layout.activity_main);
    //得到ContentResolver对象
    ContentResolver cr = getContentResolver();
    //取得电话本中开始一项的光标
    Cursor cursor = cr.query(ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
    strings = new ArrayList<String>();
    //向下移动光标
    while(cursor.moveToNext()) {
        //取得联系人名字
        int nameFieldColumnIndex = cursor.getColumnIndex(ContactsContract.PhoneLookup.DISPLAY_NAME);
        String contact = cursor.getString(nameFieldColumnIndex);
```

实例：获取手机通讯录

```
        //取得电话号码
        String contactId= cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts._ID));
        //获取电话本开始一项的Uri
        Cursor phone =  cr.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null,
        ContactsContract.CommonDataKinds.Phone.CONTACT_ID + "=" + contactId, null, null);
        while(phone.moveToNext())  {
           String phoneNumber= phone.getString(phone.getColumnIndex(
                    ContactsContract.CommonDataKinds.Phone.NUMBER));
           strings.add (contact + ": " + phoneNumber + "\n");
        }
     }
    cursor.close();
     //获取定义的ListView用来显示通讯录信息
    ListView peo_list = findViewById(R.id.show_people);
   peo_list.setAdapter(new ArrayAdapter<String>(MainActivity.this,
                                                android.R.layout.simple_list_item_1,string));
}
```

实例：获取手机通讯录

- (3) 添加权限
  - 在AndroidManifest.xml中添加联系人读写权限

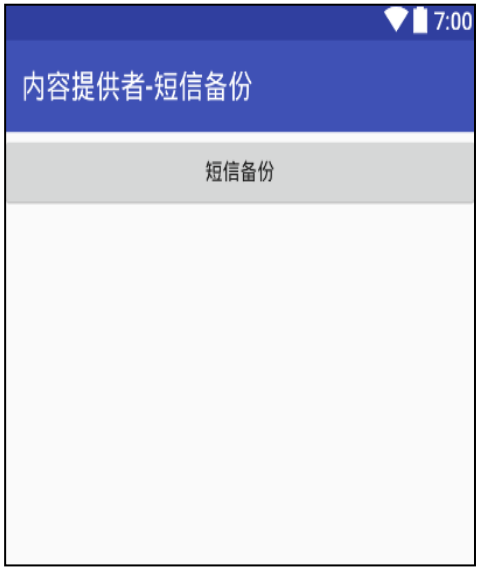
```
<uses-permission android:name="android.permission.READ_CONTACTS" >
```

  - 模拟器中长按app > 应用信息（App info） > 权限（Permissions） > 通讯录（Contacts）

实例：系统短信备份

- (1) activity_main.xml文件
  - 点击按钮获取系统短信，并保存成xml文件存放在SD卡下

```
<Button
   android:id="@+id/button1"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="短信备份"/>
```



- (2) 编写SmsInfo类
  - 封装每条短信的信息

```
class SmsInfo {
   private String address;//发送地址
   private long date;//发送时间
   private int type;//类型
   private String body;//内容
   private int id;
   public SmsInfo(String address, long date, int type, String body) {
      this.address = address;
      this.date = date;
      this.type = type;
      this.body = body;
   }
```

实例：系统短信备份

- (3)创建Sms_Xml类
  - 负责把短信信息保持成xml文件，并存放到SD卡

```
class Sms_Xml {
   public static void backupSms(List<SmsInfo> list, Context context) {
      try {
         XmlSerializer serial = Xml.newSerializer();
         File file = new File(Environment.getExternalStorageDirectory(),"mes.xml");
         FileOutputStream fi_out = new FileOutputStream(file);
         //初始化序列号器，指定xml数据写入到哪个文件以及编码
         serial.setOutput(fi_out,"utf-8");
         serial.startDocument("utf-8",true);
         //根节点
         serial.startTag(null,"smss");
         for (SmsInfo info : list){
            //构建父节点
            serial.startTag(null,"sms");
            serial.attribute(null,"id",info.getId()+"");
            //body部分
            serial.startTag(null,"body");
            serial.text(info.getBody());
            serial.endTag(null,"body");
```

```
            //address部分
            serial.startTag(null,"address");
            serial.text(info.getAddress());
            serial.endTag(null,"address");
            //type部分
            ...
            //date部分
            ...
            //父节点结束
            serial.endTag(null,"sms");
         }
      serial.endTag(null,"smss");
      serial.endDocument();
      fi_out.close();
```

实例：系统短信备份

- (4) MainActivity代码

```
public void onClick(View view) {
    //content://sms查询所有短信的uri
    Uri uri= Uri.parse("content://sms/");
    //获取ContentResolver对象
    ContentResolver contentResolver = getContentResolver();
    //通过ContentResolver对象查询系统短信
    Cursor cursor = contentResolver.query(uri, new String[]{“address”,“date”, "type","body"},null,null,null);
    List<SmsInfo> list = new ArrayList<SmsInfo>();
    while (cursor.moveToNext()) {
        String address = cursor.getString(0);
        long date = cursor.getLong(1);
        int type = cursor.getInt(2);
        String body = cursor.getString(3);
        SmsInfo smsInfo = new SmsInfo(address,date,type,body);
        list.add(smsInfo);
    }
    cursor.close();
    Sms_Xml.backupSms(list,MainActivity.this);
}
```

实例：系统短信备份

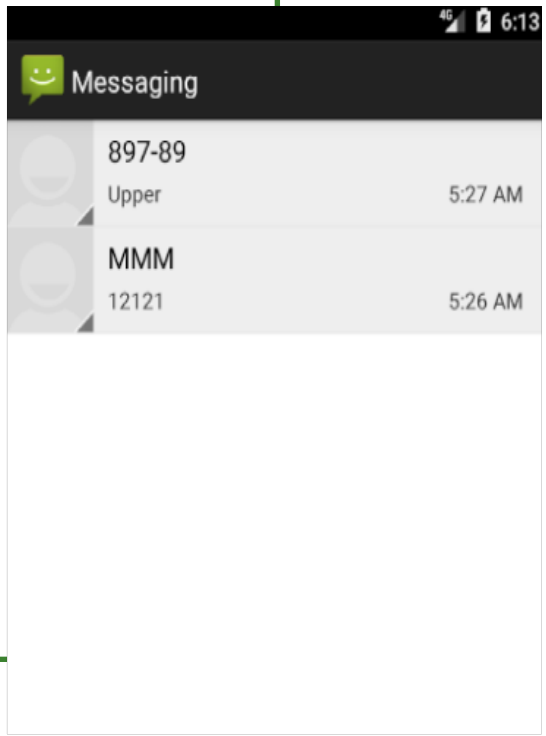
- (5) 添加权限

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

  - 模拟器中长按app > 应用信息 > 权限 > 存储空间、短信

- 运行程序
  - 模拟器工具栏中打开Extended controls，在Phone面板中发送虚拟短信

```
<?xml version='1.0' encoding='utf-8'
standalone='yes' ?>
<smss>
  <sms id="0">
    <body>Upper</body>
    <address>897-89</address>
    <type>2</type>
    <date>1500269244758</date>
  </sms>
  <sms id="0">
    <body>12121</body>
    <address>1 234-567-8</address>
    <type>2</type>
    <date>1500269178055</date>
  </sms>
</smss>
```

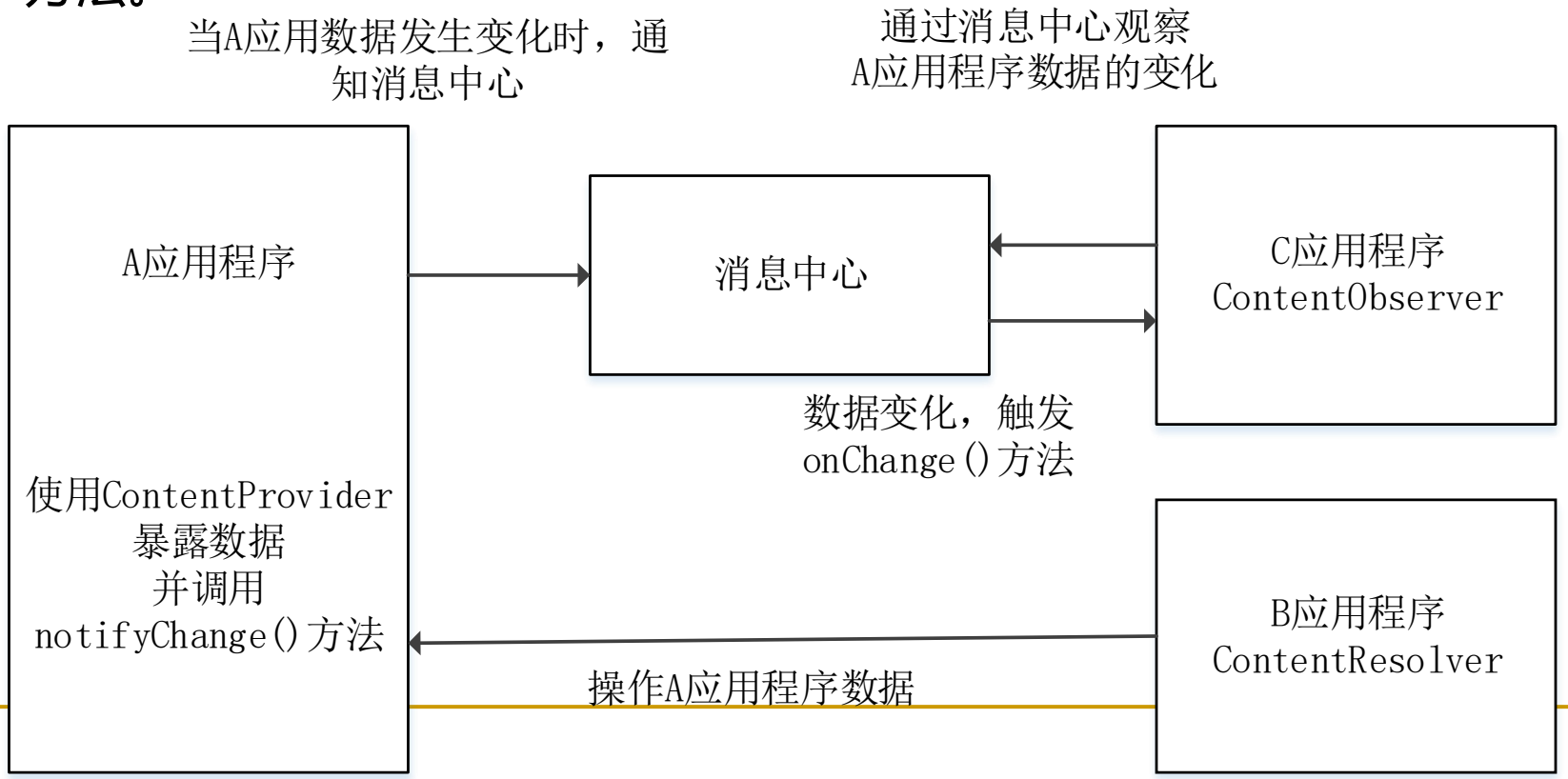


ContentObserver简介

- ContentObserver（内容观察者）用于观察(捕捉)特定Uri引起的数据库的变化，继而做一些相应的处理，类似于数据库技术中的触发器(Trigger)
- 触发器分为表触发器、行触发器，相应地ContentObserver也分为“表”ContentObserver、“行”ContentObserver，与其所监听的Uri MIME Type有关

ContentObserver简介

- 实例：A应用程序通过ContentProvider暴露自己的数据，B应用程序通过ContentResolver操作A应用程序的数据，当A应用程序的数据发生变化时，A应用程序调用notifyChange()方法向消息中心发送消息，然后C应用程序观察到数据变化时，就会触发ContentObserver的onChange()方法。



ContentObserver简介

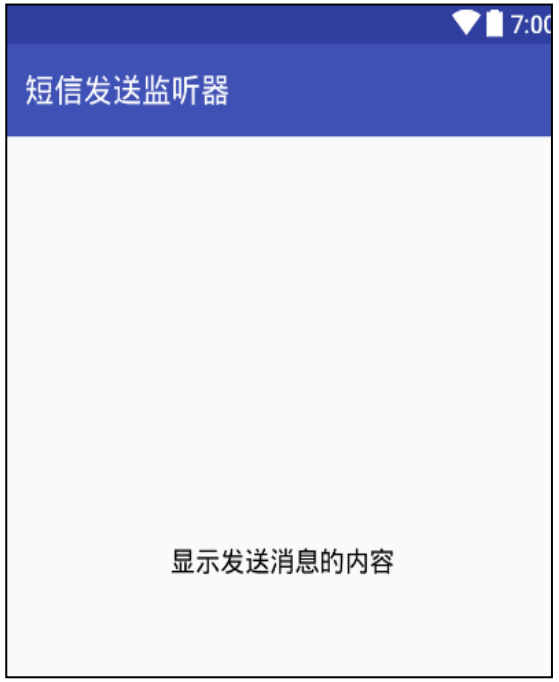
- ContentObserver类的常用操作方法

方法名称	描述
public void ContentObserver(Handler handler)	构造方法
void onChange(boolean selfChange)	观察到的Uri发生变化时，回调该方法去处理

实例：监控短信发送

- 功能
  - 通过监听Uri为content://sms的数据改变即可监听到短信数据的改变，并且在监听器的onChange(Boolean selfChange)方法查询Uri为content://sms/outbox的数据，获取用户正在发送的短信（用户正在发送的短信是保存在发件箱内）
- (1) activity_main.xml文件

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="显示发送消息的内容"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:textSize="17sp"/>
```



实例：监控短信发送

- (2) MainActivity代码

```
public class MainActivity extends Activity {
    private TextView mes_text;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mes_text=findViewById(R.id.show_mes);
        //为content://sms的数据改变注册监听器
        ContentResolver contentResolver = getContentResolver();
        Uri uri= Uri.parse("content://sms/");
        contentResolver.registerContentObserver(uri, true, new SmsObsever(new Handler()));
    }
```

实例：监控短信发送

- (2) MainActivity代码

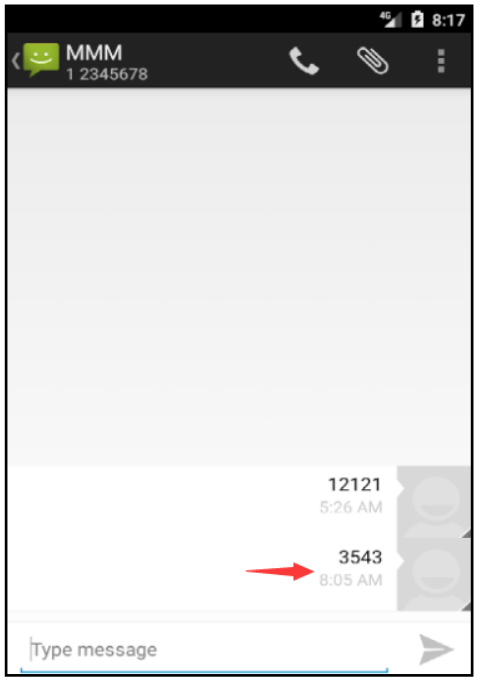
```
    //自定义的ContentObserver监听器类
    private class SmsObsever extends ContentObserver {
        public SmsObsever(Handler handler) {
            super(handler);
        }
        public void onChange(boolean selfChange) {
            //查询发件箱中的短信
            Cursor cursor = getContentResolver().query(Uri.parse("content://sms/sent"),null,null,null,null);
            //遍历查询的结果集
            while(cursor.moveToNext()) {
                String address = cursor.getString(cursor.getColumnIndex("address"));
                String body = cursor.getString(cursor.getColumnIndex("body"));
                String time = cursor.getString(cursor.getColumnIndex("date"));
                mes_text.setText("收件人："+address+"\n内容："+body+"\n发送时间："+time);
            }
            cursor.close();
        }
    }
}
```

实例：监控短信发送

- (5) 添加权限

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

- 运行程序



采用Activity来实现短信监控，必须保持该Activity不关闭。利用Android中的Service组件可以实现以后台进程方式来监听。