0
Service
①创建的方式
写一个类继承Service 重写生命周期 清单文件注册

startService onCreate->onStartCommand stopService ->onDestroy
bindService onCreate->onBind 返回值 IBinder(Binder) unBindService(ServiceConnection) ->onDestroy
ServiceConnection
onServiceConnected(ComponentName name ,IBinder)

①获取对象TelephonyManager getSystemService(TELEPHONY_SERVICE)
② 通过电话管理器对象注册一个监听 listen(PhoneStateListener, Flag);
2.1 写一个类继承PhoneStateListener 重写onCallStateChanged(int state,String incomingNumber)
MediaRecorder

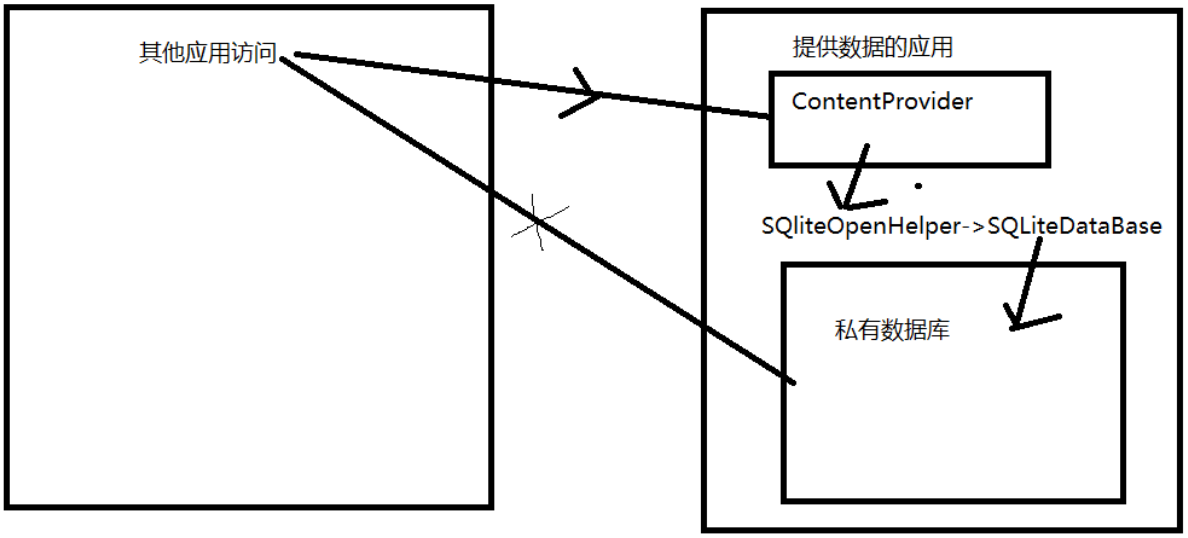
特殊的广播接收者 必须通过动态注册的方式进行注册 unregisterReceiver()

AIDL rpc 远程过程调用 ipc 进程间通信
①创建service 清单文件中声明的时候 添加一个intent-filter action
② 创建一个接口 把需要暴露的方法写在接口中 修改接口文件的扩展名 .aidl

远程调用的应用
① 通过隐式意图打开service bindService方法
② 把需要远程调用的aidl文件拷贝到当前项目中(保持aidl文件的目录结构)
③ 在onServiceConnected 使用stub.asInterface(IBinder)

进程和进程的优先级
①默认情况 一个应用使用一个进程 只有一个线程 四大组件都运行在同一个线程中 主线程
② 进程的创建和销毁 由系统统一管理 5个层级优先级
1 前台进程 2可视进程 3 服务进程 4后台进程(LRU) 5空进程

1.为什么要用内容提供者(contentProvider) contentResovler
把私有数据库暴露给其他应用使用
2 内容提供者原理



3 内容提供者实现步骤

①写一个类继承ContentProvider 重写 onCreate query delete update insert getType

② 在清单文件中声明对应的provider节点

```
<provider android:name="com.itheima.contentproviderdemo.MyProvider"
          android:authorities="com.itheima.provider"
          android:exported="true"></provider>
```

authorities = 通过这个字符串来决定访问的是哪个内容提供者 高版本的设备 还需要配置一个参数 exported = true;

③ 通过URIMatch添加URi的匹配规则

Pub com.itheima.provider: com.itheima.contentproviderdemo.MyProvider

```
<provider
android:name="com.itheima.contentproviderdemo.MyProvider"
   android:authorities="com.itheima.provider"
android:exported="true">
</provider>

private static final UriMatcher sURIMatcher = new
UriMatcher(UriMatcher.NO_MATCH);
    private static final int QUERY_SUCESS = 0;
    private static final int INSERT_MATCHED = 1;
    private static final int UPDATE_MATCHED = 2;
    private static final int DELETE_MATCHED = 3;
    static
    {
        //给当前的URI匹配器添加一个匹配规则
        sURIMatcher.addURI("com.itheima.provider",
"query", QUERY_SUCESS);
        sURIMatcher.addURI("com.itheima.provider",
"insert", INSERT_MATCHED);
        sURIMatcher.addURI("com.itheima.provider",
"update", UPDATE_MATCHED);
        sURIMatcher.addURI("com.itheima.provider",
"delete", DELETE_MATCHED);
    }
```

Content://com.itheima.provider/query

```
ContentResolver contentResolver =
getContentResolver();
        Uri uri =Uri.parse
("content://com.itheima.provider/query");
    Cursor cursor = contentResolver.query(uri,
null, null, null, null);
```

URIMatcher 匹配上了 query的规则 就会返回 QUREY_SUCESS

4 备份短信案例

5 通过内容提供者插入短信

通过contentresolver 访问短信的contentprovider

确定URI 通过查询短信provider的源码

```
<provider android:name="SmsProvider"
          android:authorities="sms"
          android:multiprocess="true"
          android:readPermission="android.permission.READ_SMS"
          android:writePermission="android.permission.WRITE_SMS" />
```

所以可以确定访问短信的provider 需要使用的authorities 是 sms

同时发现需要一个读的权限 如果需要操作短信的provider写入数据还需要写的权限

uri "content://sms"

authorities 确定之后 还需要知道子路径如何写

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

static {
    sURLMatcher.addURI("sms", null, SMS_ALL);
    sURLMatcher.addURI("sms", "#", SMS_ALL_ID);
```

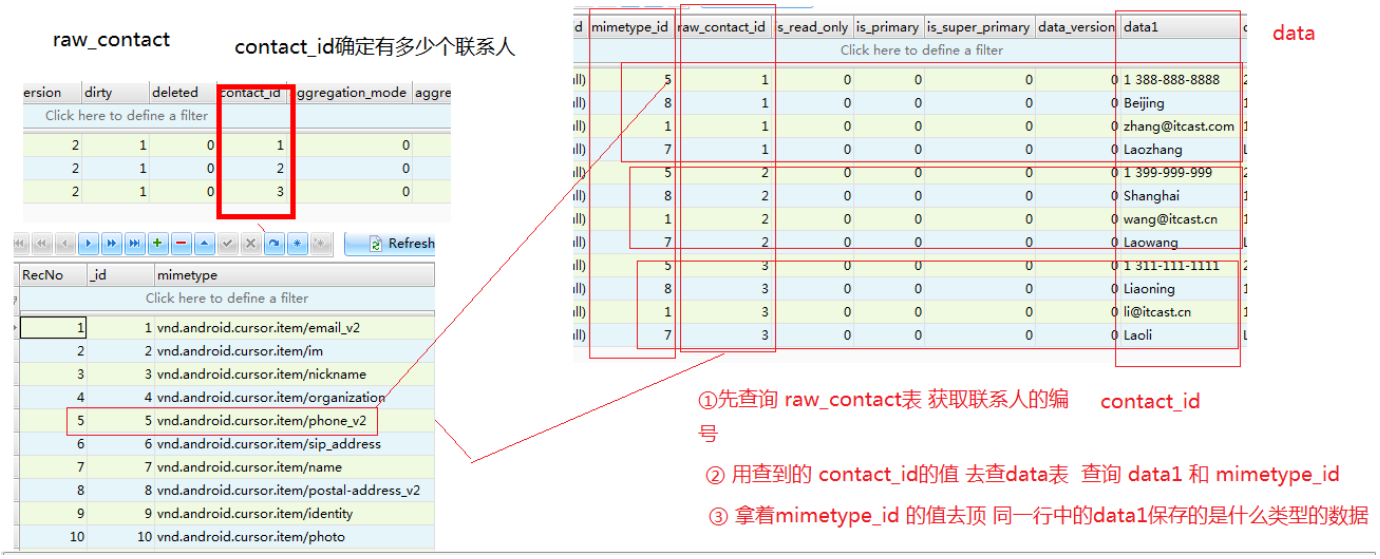
content://sms 通过这个uri就可以访问到所有的短信内容

```
1.  public void querySms(View v){
2.          //获取内容解析者
3.          ContentResolver resolver = getContentResolver();
4.          //通过查看smsProvider的代码发现 子路径传null 代表查询所有的短信
5.          Uri uri = Uri.parse("content://sms");
6.          String[] projection = {"address","date","body"};
7.          Cursor cursor = resolver.query(uri, projection, null, null, null);
8.          while(cursor.moveToNext()){
9.              Sms sms = new Sms();
10.             String address = cursor.getString(0);
11.             String date = cursor.getString(1);
12.             String body = cursor.getString(2);
13.             sms.address = address;
14.             sms.body = body;
15.             sms.date = date;
16.
17.             smsList.add(sms);
18.         }
19.         for(Sms sms:smsList){
20.             System.out.println(sms);
21.         }
22.     }
```

插入数据

```
public void insert(View v){
        //获取contentResolver
        ContentResolver resolver = getContentResolver();
        Uri url = Uri.parse("content://sms");
        ContentValues values = new ContentValues();
        values.put("date", System.currentTimeMillis());
        values.put("address", "95555");
        values.put("body", "王晓冬先生:您的尾号为8888的金卡账户,信用卡本月账单还有68,123.00没换,请注意账户资金变化.....");
        resolver.insert(url, values);
    }
```

6 读取联系人案例



7 插入联系人案例

```
public class MainActivity extends Activity {

    private EditText et_address;
    private EditText et_name;
    private EditText et_email;
    private EditText et_phone;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //找到控件
```

```
13.
14.         et_address = (EditText) findViewById(R.id.et_address);
15.         et_name = (EditText) findViewById(R.id.et_name);
16.         et_email = (EditText) findViewById(R.id.et_email);
17.         et_phone = (EditText) findViewById(R.id.et_phone);
18.     }
19.
20.
21.    public void save(View v){
22.         //①获取内容解析者
23.         ContentResolver resolver = getContentResolver();
24.         //②获取URI
25.         Uri raw_contact_uri = Uri.parse("content://com.android.contacts/raw_contact
    s");
26.         Uri data_uri = Uri.parse("content://com.android.contacts/data");
27.         //③确定插入到 raw_contact表中id的值
28.         Cursor cursor = resolver.query(raw_contact_uri, new String[]{"contact_id"},
     null, null, null);
29.         //获取当前查询的结果一共有多少行 那么 就用查询结果加一 值作为新插入的contac
    t_id的值
30.         int count = cursor.getCount();
31.         //④拿到确定的contact_id值之后 把对应的值 插入到 raw_contact表中
32.         ContentValues values = new ContentValues();
33.         values.put("contact_id", count+1);
34.         resolver.insert(raw_contact_uri, values);
35.         //⑤操作data表 插入对应的内容   要保存联系人的 姓名 电话 地址 邮箱 每一个数
    据都对应这个data表中的一行
36.         ContentValues add_values = new ContentValues();
37.         //获取用户输入的地址
38.         String address = et_address.getText().toString();
39.         //添加要插入列名和具体的值
40.         add_values.put("raw_contact_id", count+1);
41.         //data1 保存的数据 具体的值
42.         add_values.put("data1", address);
43.         //mimetype指定了当前存入数据的类型
44.         add_values.put("mimetype", "vnd.android.cursor.item/postal-address_v2");
45.         resolver.insert(data_uri, add_values);
46.
47.         //保存了姓名
48.         ContentValues name_values = new ContentValues();
49.         String name = et_name.getText().toString();
50.         name_values.put("raw_contact_id", count+1);
51.         name_values.put("data1", name);
52.         name_values.put("mimetype", "vnd.android.cursor.item/name");
53.         resolver.insert(data_uri, name_values);
54.         //insert into info values(null)
55.
56.         //保存了电话
57.         ContentValues phone_values = new ContentValues();
58.         String phone = et_phone.getText().toString();
59.         phone_values.put("raw_contact_id", count+1);
60.         phone_values.put("data1", phone);
61.         phone_values.put("mimetype", "vnd.android.cursor.item/phone_v2");
62.         resolver.insert(data_uri, phone_values);
```

```
63.
64.         //保存了email
65.         ContentValues email_values = new ContentValues();
66.         String email = et_email.getText().toString();
67.         email_values.put("raw_contact_id", count+1);
68.         email_values.put("data1", email);
69.         email_values.put("mimetype", "vnd.android.cursor.item/email_v2");
70.         resolver.insert(data_uri, email_values);
71.     }
72.
73. }
```

8 内容观察者 contentOberserver

当数据发生改变的时候可以通过内容解析者发出一个通知 contentresolver.notifychanged
通过内容解析者还可以注册一个内容观察者

```
1.  public class MainActivity extends Activity {
2.
3.      @Override
4.      protected void onCreate(Bundle savedInstanceState) {
5.          super.onCreate(savedInstanceState);
6.          setContentView(R.layout.activity_main);
7.          //获取内容解析者
8.          ContentResolver resolver = getContentResolver();
9.          Uri uri = Uri.parse("content://com.itheima.provider");
10.         MyObserver observer = new MyObserver(new Handler());
11.         //第二个参数 如果传入true 只要uri前面的部分匹配上了 就可以收到通知
12.         //如果是false 只有整个URI都匹配上 才能收到通知
13.         resolver.registerContentObserver(uri, false, observer);
14.     }
15.
16.
17.    private class MyObserver extends ContentObserver{
18.
19.     public MyObserver(Handler handler) {
20.         super(handler);
21.         // TODO Auto-generated constructor stub
22.     }
23.        @Override
24.     public void onChange(boolean selfChange, Uri uri) {
25.            System.out.println(uri);
26.     }
27.    }
28.
29. }
```

9 内容观察者应用场景

今日回顾
contentprovider 内容提供者 ☆☆☆☆☆
内容提供者 开发过程
authorities:"com.itheima.provider"
exported true
Uri匹配器
URIMatcher

通过内容提供者访问数据 访问数据库用到的api SQliteDatabase
使用内容提供者把自己的数据库暴露给别人

contentresolver 内容解析者 ☆☆☆☆☆
//①获取内容解析者对象
//②确定uri content://authorities/子路径
//③ 调用insert delete update query

操作联系人 短信的数据库
contentObserver 内容观察者 ☆☆☆

activity 界面
生命周期
onCreate 初始化的操作
①界面的初始化
②数据的初始化 文件 sp 网络(联网开线程 子线程不能跟新UI handler) 上一个activity(intent) 数据库
③给按钮添加点击事件
setonclicklistener setOnItemClickListener()
④操作其他组件 注册一个广播接收者 开启一个服务
registerReceiver bindservice startService
onStart 执行完onStart 可见不可操作
onResume 执行完onResume 可见并且可以操作 加载数据

onPause 可见不可操作
onstop 不可见不可操作 刷新界面的操作停止
onDestroy 关闭数据库 注销广播接收者 unbindService

onRestart

service 没有界面的activty 可以把长期运行的操作放到service中
MediaPlayer

onCreate onStartCommand onBind onDestroy

startService
bindService

aidl rpc ipc

broadcastreceiver 可以不通过清单文件注册
系统广播
自定义的广播
sendbroacast 无序广播 不能中断 不能修改
sendorderedbroadcast 有序广播 可以中断 可以修改

contentprovider

IPC 进程间通信

四大组件都可以进行进程间通信
Activity 隐式意图
Service 隐式意图
BroadCastReceiver intent

contentprovider contentResolver访问内容提供者 访问私有数据库