0.回顾
服务作用: 做一些 需要长期运行 但还不要跟用户交互的任务 ②可以提升进程的优先级 避免进程被系统回收
startservice oncreate->onstartcommand(可以执行多次) stopservice
bindservice onCreate->onbind (只执行一次) unbindservice

bindservice获取service中的内部类对象 并且通过这个对象调用service的方法
bindservice(context, ServiceConnection,int flag)

ServiceConnection中有两个待实现的方法
onServiceConnected(,IBinder service)
Service中onbind方法的返回值
onServiceDisconnected

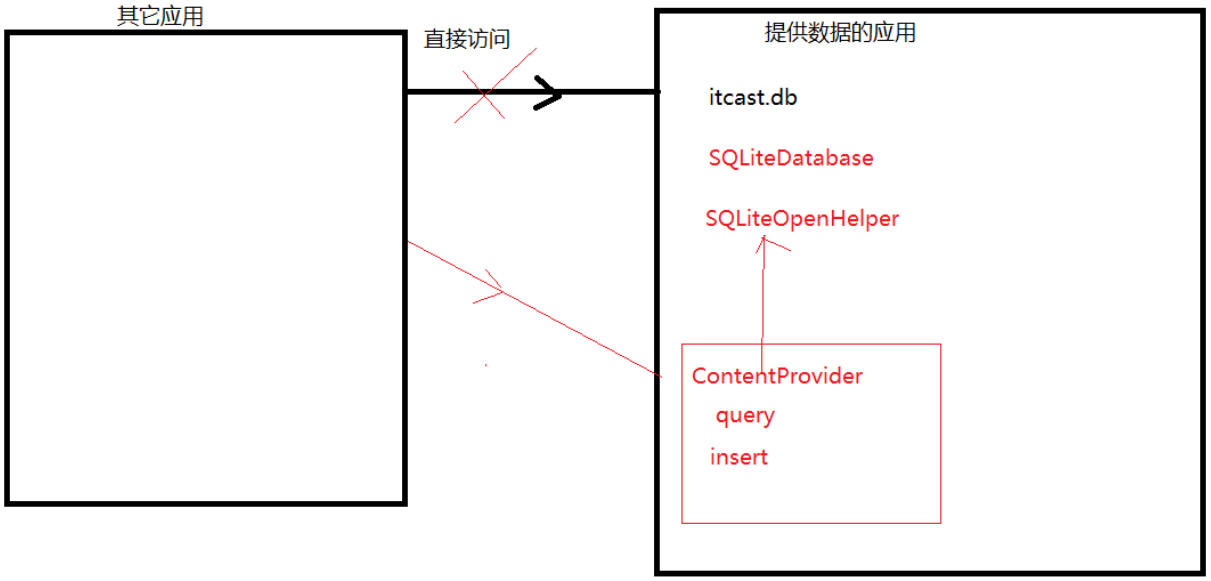
AIDL 跨进程调用方法
IService. interface
IService.java-> IService.aidl (要把public 去掉)
MyBinder extend Binder implement IService
MyBinder extend Stub

ContentProvider
ContentResolver
ContentObserver

1.为什么要用内容提供者(contentProvider)

跨应用提供数据 让其他应用访问本应用数据库中的内容

2 内容提供者原理



内容提供者作用: 在不同应用之间 共享数据库中的数据
访问内容提供者提供的数据 要使用contentresovler 内容解析者

3 内容提供者实现步骤

四大组件都是一个套路
① 创建一个类 继承ContentProvider 重写里面方法
② 在清单文件中注册相应provider 必须指定authorities 属性 还要添加一个属性 exported = true

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

③ 在provider中处理uri匹配相关内容
3.1 创建URI匹配器 在provider 中搞静态代码块 在static代码块中添加 uri匹配的规则

```
private MyOpenHelper openHelper;
    private static final UriMatcher sURIMatcher = new UriMatcher(UriMatcher.NO_MAT
CH);
    private static final int MATCH_UPDATE = 0;
    private static final int MATCH_QUERY = 1;
    static{
        //通过uri匹配器 添加匹配的路径规则 cn.itcast.provider/query 如果调用match
方法匹配上了这个规则 那么就会返回MATCH_UPDATE
        //如果传入的uri没有匹配任何预先加入的uri 就会返回NO_MATCH
        sURIMatcher.addURI("cn.itcast.provider", "query", MATCH_QUERY);
    }
```

3.2 根据业务逻辑 在不同的数据库操作方法中 处理uri匹配的流程

```
    //第一个参数 就是用来匹配具体路径 决定是否让对方访问相应方法
        int result = sURIMatcher.match(uri);
        if(result == MATCH_QUERY){
            SQLiteDatabase db = openHelper.getReadableDatabase();
            Cursor cursor = db.query("info", projection, selection, selectionArgs,
```

```
 null, null, sortOrder);
            return cursor;
        }else{
            throw new IllegalStateException("口令不正确");
        }
```

I/ActivityThread(1820): Pub cn.itcast.provider: cn.itcast.contentproviderdemo.MyProvider

④ 其他应用访问contentprovider方法

4.1 获得contentresolver对象

```
ContentResolver resolver = getContentResolver();
```

4.2 通过contentresolver调用相关方法访问contentprovider

需要注意 uri 要以content://开头 具体的路径 要跟contentprovider中 定义的uri匹配规则匹配上

```
   Uri uri = Uri.parse("content://cn.itcast.provider/query");
    Cursor cursor = resolver.query(uri, null, null, null, null);
```

contentprovider代码

```
public class MyProvider extends ContentProvider {

    //URI 统一资源标识符  url 子父类 URI 爹  url儿子 http:// ftp:// https://
    //URI 可以自定义协议  cn.itcast.provider/update1
    private MyOpenHelper openHelper;
    private static final UriMatcher sURIMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    private static final int MATCH_UPDATE = 0;
    private static final int MATCH_QUERY = 1;
    private static final int MATCH_INSERT = 0;
    private static final int MATCH_DELETE = 0;
    static{
        //通过uri匹配器 添加匹配的路径规则 cn.itcast.provider/query 如果调用match方法匹配上了这个规则 那么就会返回MATCH_UPDATE
        //如果传入的uri没有匹配任何预先加入的uri 就会返回NO_MATCH
        sURIMatcher.addURI("cn.itcast.provider", "query", MATCH_QUERY);
        sURIMatcher.addURI("cn.itcast.provider", "update", MATCH_UPDATE);
        sURIMatcher.addURI("cn.itcast.provider", "insert", MATCH_INSERT);
        sURIMatcher.addURI("cn.itcast.provider", "delete", MATCH_DELETE);
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        int result = sURIMatcher.match(uri);
        if(result == MATCH_DELETE){
```

```
            SQLiteDatabase db = openHelper.getReadableDatabase();
            int delete = db.delete("info", selection, selectionArgs);
            return delete;
        }else{
            return 0;
        }

    }

    @Override
    public String getType(Uri uri) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        int result = sURIMatcher.match(uri);
        if(result==MATCH_INSERT){
            SQLiteDatabase db = openHelper.getReadableDatabase();
            long insert = db.insert("info", null, values);
            //通过内容解析者 发送通知 告知内容发生变化 第一个参数 uri 通过这个uri确定是哪个内容提供者对应数据发生了改变
            //第二个 参数 ContentObserver 如果传入了一个内容观察者对象 那么 只有这个内容观察者能收到变化的消息
            //如果传了null 只要观察着第一个参数传入的uri的内容观察者都能收到变化的消息
            getContext().getContentResolver().notifyChange(uri, null);
            return Uri.parse(String.valueOf(insert));
        }else{
            return null;
        }
    }

    @Override
    public boolean onCreate() {
        openHelper = new MyOpenHelper(getContext());
        return false;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        //第一个参数 就是用来匹配具体路径 决定是否让对方访问相应方法
        int result = sURIMatcher.match(uri);
        if(result == MATCH_QUERY){
            SQLiteDatabase db = openHelper.getReadableDatabase();
            Cursor cursor = db.query("info", projection, selection, selectionArgs, null, null, sortOrder);
            return cursor;
        }else{
            throw new IllegalStateException("口令不正确");
        }

```

```
74.
75.     }
76.
77.     @Override
78.     public int update(Uri uri, ContentValues values, String selection,
79.             String[] selectionArgs) {
80.         int result = sURIMatcher.match(uri);
81.         if(result == MATCH_UPDATE){
82.             SQLiteDatabase db = openHelper.getReadableDatabase();
83.             int update = db.update("info", values, selection, selectionArgs);
84.             return update;
85.         }else{
86.             return 0;
87.         }
88.     }
89. }
```

OpenHelper代码

```
1.  public class MyOpenHelper extends SQLiteOpenHelper {
2.
3.      public MyOpenHelper(Context context) {
4.          super(context, "itcast.db", null, 1);
5.          // TODO Auto-generated constructor stub
6.      }
7.
8.      @Override
9.      public void onCreate(SQLiteDatabase db) {
10.         db.execSQL("create table info(_id integer primary key autoincrement,name v
    archar(20),phone varchar(20))");
11.         db.execSQL("insert into info(name,phone) values('王五','13777777')");
12.         db.execSQL("insert into info(name,phone) values('赵四','13888888')");
13.     }
14.
15.     @Override
16.     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
17.         // TODO Auto-generated method stub
18.
19.     }
20.
21. }
```

清单文件代码

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.      package="cn.itcast.contentproviderdemo"
4.      android:versionCode="1"
5.      android:versionName="1.0" >
6.
7.      <uses-sdk
8.          android:minSdkVersion="8"
9.          android:targetSdkVersion="17" />
```

```
10.
11.     <application
12.         android:allowBackup="true"
13.         android:icon="@drawable/ic_launcher"
14.         android:label="@string/app_name"
15.         android:theme="@style/AppTheme" >
16.         <activity
17.             android:name="cn.itcast.contentproviderdemo.MainActivity"
18.             android:label="@string/app_name" >
19.             <intent-filter>
20.                 <action android:name="android.intent.action.MAIN" />
21.
22.                 <category android:name="android.intent.category.LAUNCHER" />
23.             </intent-filter>
24.         </activity>
25.         <provider android:name="cn.itcast.contentproviderdemo.MyProvider"
26.             android:authorities="cn.itcast.provider"
27.             android:exported="true"></provider>
28.     </application>
29.
30. </manifest>
```

4 备份短信案例

思路 ① 获取要访问的短信表的uri路径的authorities

查询TelephonyProvider的清单文件 获得provider的 authorities的值 是sms 所以 uri要以content://sms开头

操作短信的内容提供者需要权限

android.permission.READ_SMS

android.permission.WRITE_SMS

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

② 要获取访问短信表的路径如何写 查询SmsProvider的urimatch相应方法

```
716     private static final int SMS_STATUS_ID = 20;
717     private static final int SMS_STATUS_PENDING = 21;
718     private static final int SMS_ALL_ICC = 22;
719     private static final int SMS_ICC = 23;
720     private static final int SMS_FAILED = 24;
721     private static final int SMS_FAILED_ID = 25;
722     private static final int SMS_QUEUED = 26;
723     private static final int SMS_UNDELIVERED = 27;
724
725     private static final UriMatcher sURLMatcher =
726             new UriMatcher(UriMatcher.NO_MATCH);
727
728     static {
729         sURLMatcher.addURI("sms", null, SMS_ALL);
730         sURLMatcher.addURI("sms", "#", SMS_ALL_ID);
731         sURLMatcher.addURI("sms", "inbox", SMS_INBOX);
732         sURLMatcher.addURI("sms", "inbox/#", SMS_INBOX_ID);
733         sURLMatcher.addURI("sms", "sent", SMS_SENT);
734         sURLMatcher.addURI("sms", "sent/#", SMS_SENT_ID);
```

发现 子路径传null 就可以访问到所有的短信

所以 具体访问到SmsProvider的uri 就是 content://sms

确定了uri之后 就可以通过contentresolver执行query insert update delete操作了

```
1.  public class MainActivity extends Activity {
2.      private ArrayList<Sms> sms_list = new ArrayList<Sms>();
3.
4.      @Override
5.      protected void onCreate(Bundle savedInstanceState) {
6.          super.onCreate(savedInstanceState);
7.          setContentView(R.layout.activity_main);
8.      }
9.
10.
11.    public void querysms(View v){
12.        //清空集合
13.        sms_list.clear();
14.        //获取contentresolver  内容解析者
15.        ContentResolver resolver = getContentResolver();
16.        Uri uri = Uri.parse("content://sms");
17.        //通过内容解析者 查询短息数据库 获取 date(短信收到的时间) address(短信发送
    人) body(短信内容)
18.     Cursor cursor = resolver.query(uri, new String[]{"date","address","body"}, nul
    l, null, null);
19.        while (cursor.moveToNext()) {
20.         String date = cursor.getString(0);
21.         String address = cursor.getString(1);
22.         String body = cursor.getString(2);
23.         Sms sms = new Sms();
24.         sms.date = date;
25.         sms.address = address;
```

```
            sms.body = body;
            sms_list.add(sms);
            System.out.println(sms);
        }
       }

       public void savesms(View v){
           //获取xml序列化器对象
           XmlSerializer serializer = Xml.newSerializer();
           FileOutputStream fos;
        try {
            fos = openFileOutput("sms.xml", MODE_PRIVATE);
            serializer.setOutput(fos, "utf-8");
            //先写开始标记<?xml version="1.0" encoding="utf-8"?> standalone true说明不依赖其他文档
            serializer.startDocument("utf-8", true);
            //开始写<SmsList>标签  整个文档的根标签
            serializer.startTag(null, "SmsList");
            for(Sms sms:sms_list){
                //写<Sms>
                serializer.startTag(null, "Sms");

                //写address节点
                serializer.startTag(null, "address");
                serializer.text(sms.address);
                serializer.endTag(null, "address");

               //写body节点
                serializer.startTag(null, "body");
                serializer.text(sms.body);
                serializer.endTag(null, "body");

               //写date节点
                serializer.startTag(null, "date");
                serializer.text(sms.date);
                serializer.endTag(null, "date");

                //写</Sms>
                serializer.endTag(null, "Sms");
            }

           //写</SmsList>标签
            serializer.endTag(null, "SmsList");
            serializer.endDocument();
            fos.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

       }
}
```

5 通过内容提供者插入短信

```
1.  public void insertsms(View v){
2.        //获取内容解析者 contentresolver
3.        ContentResolver resolver = getContentResolver();
4.        Uri url = Uri.parse("content://sms");
5.     ContentValues values = new ContentValues();
6.     values.put("address", 95555);
7.     values.put("date", System.currentTimeMillis());
8.     values.put("body", "李胜文先生:您的尾号为8888的金葵花信用卡,本期账单尚余68,987
   .00未还,请及时还款,以免影响您的信用记录.");
9.     resolver.insert(url, values);
10.   }
```

注意: 插入短信数据在4.4之后 不能使用 只有默认的短信应用才能修改短信数据库

6 读取联系人案例

跟联系人数据查询相关的表有三张表

data 保存了联系人相关的所有记录 但是 每一条信息对应一条记录 并不是一个联系人对应一条记录

raw_contacts 每创建一个联系人就会在这张表中生成一条记录 contact_id

mime_types 记录了数据的媒体类型

操作联系人的内容提供者需要权限

android.permission.READ_CONTACTS

android.permission.WRITE_CONTACTS

具体思路 首先要获取操作data表和 raw_contacts表的uri 通过查询ContactsProvider的清单文件发现

```
20      <provider android:name="ContactsProvider2"
21          android:authorities="contacts;com.android.contacts"
22          android:label="@string/provider_label"
23          android:multiprocess="false"
24          android:readPermission="android.permission.READ_CONTACTS"
25          android:writePermission="android.permission.WRITE_CONTACTS">
26          <path-permission
27                  android:pathPrefix="/search_suggest_query"
28                  android:readPermission="android.permission.GLOBAL_SEARCH" />
29          <path-permission
30                  android:pathPrefix="/search_suggest_shortcut"
31                  android:readPermission="android.permission.GLOBAL_SEARCH" />
32          <path-permission
33                  android:pathPattern="/contacts/.*/photo"
34                  android:readPermission="android.permission.GLOBAL_SEARCH" />
35          <grant-uri-permission android:pathPattern=".*" />
36      </provider>
```

所以 uri 是以 content://com.android.contacts开头

具体访问某个表的uri还需要通过查询具体代码发现

```
final UriMatcher matcher = sUriMatcher;
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, "contacts/#/suggestions",
        AGGREGATION_SUGGESTIONS);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/#/suggestions/*",
        AGGREGATION_SUGGESTIONS);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/#/photo", CONTACTS_PHOTO);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/filter/*", CONTACTS_FILTER);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/lookup/*", CONTACTS_LOOKUP);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/lookup/*/#", CONTACTS_LOOKUP_ID);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/as_vcard/*", CONTACTS_AS_VCARD);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/as_multi_vcard/*",
        CONTACTS_AS_MULTI_VCARD);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/strequent/", CONTACTS_STREQUENT);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/strequent/filter/*",
        CONTACTS_STREQUENT_FILTER);
matcher.addURI(ContactsContract.AUTHORITY, "contacts/group/*", CONTACTS_GROUP);

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);
matcher.addURI(ContactsContract.AUTHORITY, "raw_contacts/#/entity", RAW_CONTACT_ENTITY_ID);

matcher.addURI(ContactsContract.AUTHORITY, "raw_contact_entities", RAW_CONTACT_ENTITIES);

matcher.addURI(ContactsContract.AUTHORITY, "data", DATA);
matcher.addURI(ContactsContract.AUTHORITY, "data/#", DATA_ID);
```

通过查询代码 确定 操作raw_contacts表路径写成

content://com.android.contacts/raw_contacts

查询data表 uri写成

content://com.android.contacts/data

```
/**
     * 查询联系人表返回集合
     * @param context
     * @return
     */
    public static ArrayList<Contact> getContacts(Context context){
        ArrayList<Contact> contact_list = new ArrayList<Contact>();
         ContentResolver resolver = context.getContentResolver();
           //raw_contacts表对应的uri
           Uri uri= Uri.parse("content://com.android.contacts/raw_contacts");
           //data表对应的uri
           Uri data_uri= Uri.parse("content://com.android.contacts/data");
           //通过内容解析者查询 raw_contacts表 获取 contact_id值
        Cursor cursor = resolver.query(uri, new String[]{"contact_id"}, null, null
, null);
        while (cursor.moveToNext()) {
            //每一个contact_id值对应一组联系人数据
            String contact_id = cursor.getString(0);
            if(contact_id == null){
                continue;
            }
```

```
            System.out.println("contact_id="+contact_id);

            String selection = "raw_contact_id =?";
            String[] selectionArgs = new String[]{contact_id};
//          Cursor cursor2 = resolver.query(data_uri, null, selection, selectionArgs, null);
//          for( int i = 0;i<cursor2.getColumnCount();i++){
//              String name = cursor2.getColumnName(i);
//              if(name.startsWith("mime")){
//                  System.out.println(name);
//              }
//          }
            //创建联系人对象
            Contact contact = new Contact();
            //拿着查询到的contact_id 值去data表中查询对应的联系人数据
            Cursor data_cursor = resolver.query(data_uri, new String[]{"mimetype", "data1"}, selection, selectionArgs, null);
            while (data_cursor.moveToNext()) {
                //如果mimetype 返回是email_v2 说明data中保存的是email
                String mimetype = data_cursor.getString(0);
                String data = data_cursor.getString(1);
                if("vnd.android.cursor.item/email_v2".equals(mimetype)){
                    contact.email = data;
                }else if("vnd.android.cursor.item/name".equals(mimetype)){
                    contact.name = data;
                }else if("vnd.android.cursor.item/postal-address_v2".equals(mimetype)){
                    contact.address = data;
                }else if("vnd.android.cursor.item/phone_v2".equals(mimetype)){
                    contact.phone = data;
                }
                System.out.println("mimetype="+mimetype+"====data="+data);
            }
            //把联系人对象添加到集合中
            contact_list.add(contact);
        }
        return contact_list;
    }
```

7 插入联系人案例

思路 先插入数据到raw_contacts表 在插入数据到data表

```
public class MainActivity extends Activity {

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

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et_address = (EditText) findViewById(R.id.et_address);
        et_email = (EditText) findViewById(R.id.et_email);
        et_name = (EditText) findViewById(R.id.et_name);
        et_phone = (EditText) findViewById(R.id.et_phone);
    }

    public void save(View v) {
        // 内容解析者 contentresolver
        ContentResolver resolver = getContentResolver();
        // raw_contacts表对应的uri
        Uri uri = Uri.parse("content://com.android.contacts/raw_contacts");
        // data表对应的uri
        Uri data_uri = Uri.parse("content://com.android.contacts/data");
        // 先确定要插入到raw_contacts contact_id的值
        // 首先查询表 看有多少个返回值 把返回值数目+1 就是要插入的具体数据
        Cursor cursor = resolver.query(uri, new String[] { "contact_id" },
                null, null, null);
        // 获取cursor返回的行数目 +1
        int count = cursor.getCount() + 1;
        ContentValues contact_id = new ContentValues();
        contact_id.put("contact_id", count);
        resolver.insert(uri, contact_id);

        // 插入跟姓名相关的一行记录
        ContentValues name_values = new ContentValues();
        name_values.put("raw_contact_id", count);
        name_values.put("data1", et_name.getText().toString().trim());
        name_values.put("mimetype", "vnd.android.cursor.item/name");
        // 操作data表
        resolver.insert(data_uri, name_values);

        // 插入跟电话相关的一行记录
        ContentValues phone_values = new ContentValues();
        phone_values.put("raw_contact_id", count);
        phone_values.put("data1", et_phone.getText().toString().trim());
        phone_values.put("mimetype", "vnd.android.cursor.item/phone_v2");
        // 操作data表
        resolver.insert(data_uri, phone_values);

        // 插入email相关的一行记录
        ContentValues email_values = new ContentValues();
        email_values.put("raw_contact_id", count);
        email_values.put("data1", et_email.getText().toString().trim());
        email_values.put("mimetype", "vnd.android.cursor.item/email_v2");
        // 操作data表
        resolver.insert(data_uri, email_values);

        // 插入跟姓名相关的一行记录
        ContentValues address_values = new ContentValues();
        address_values.put("raw_contact_id", count);
        address_values.put("data1", et_address.getText().toString().trim());
```

```
        address_values.put("mimetype", "vnd.android.cursor.item/postal-address_v2");
        // 操作data表
        resolver.insert(data_uri, address_values);
    }

}
```

8 内容观察者

contentobserver

要在数据库数据发生变化的时候 通过内容解析者 发送一个notifychanged消息

```
public Uri insert(Uri uri, ContentValues values) {
        int result = sURIMatcher.match(uri);
        if(result==MATCH_INSERT){
            SQLiteDatabase db = openHelper.getReadableDatabase();
            long insert = db.insert("info", null, values);
            //通过内容解析者 发送通知 告知内容发生变化 第一个参数 uri 通过这个uri确定是哪个内容提供者对应数据发生了改变
            //第二个 参数 ContentObserver 如果传入了一个内容观察者对象 那么 只有这个内容观察者能收到变化的消息
            //如果传了null 只要观察着第一个参数传入的uri的内容观察者都能收到变化的消息
            getContext().getContentResolver().notifyChange(uri, null);
            return Uri.parse(String.valueOf(insert));
        }else{
            return null;
        }
    }
```

在要观察数据变化的地方 通过内容解析者注册一个内容观察者

contentobserver 是抽象类 要继承并重写方法onChange()

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Uri uri = Uri.parse("content://cn.itcast.provider");
        MyObserver observer = new MyObserver(new Handler());
//获取内容解析者 ,通过内容解析者注册内容观察者
//第一个参数 uri 就是要接收变化消息的uri  要跟notifyChange(uri,null) 第一个参数对应
//第二个参数 路径的匹配方式 如果是true 只要前面的authoritese部分匹配上了就可以收到变化消息
//如果是false 只有整个路径都匹配了才能收到消息
// uri是  content://cn.itcast.provider  第二个参数穿了false 那么 它不能跟  content://cn.itcast.provider/insert匹配
```

```
14.  //第三个参数 就是要注册的内容观察者
15.          getContentResolver().registerContentObserver(uri, true, observer);
16.      }
17.
18.
19.     private class MyObserver extends ContentObserver{
20.
21.      public MyObserver(Handler handler) {
22.          super(handler);
23.          // TODO Auto-generated constructor stub
24.      }
25.      @Override
26.      public void onChange(boolean selfChange, Uri uri) {
27.          System.out.println("uri = "+uri+"变化了");
28.          //收到数据库内容变化的通知 在界面上刷新 展示最新的数据
29.      }
30.     }
31.  }
```

9 内容观察者应用场景

比如 聊天应用 收到服务端传来的消息 先把消息保存到收据库 当数据库内容发生改变的时候就可以先通过内容解析者发送消息

在需要展示聊天界面的activity里注册内容观察者 当内容观察者收到数据库发生改变的消息时 可以重新查询消息记录的数据库,刷新整个界面,展示最新的聊天记录

10 getApplicationContext 和 activity context

getApplicationContext 获取到的是应用的上下文 生命周期跟应用相同 使用getApplicationContext获取到的上下文不会造成内存泄露

只有在创建alertdialog的时候 必须使用activity作为上下文 因为 alertdialog要显示在activity的上面 必须传一个activity的上下文给它才能够正确显示

吐司 跟alertdialog 不同 吐司是系统级的显示控件 会显示在所有应用的最上层 不会被其他界面遮挡 所以 传递应用的上下文也可以显示吐司

activity >service>broadcastreceiver >contentprovider contentresolver

configchange screensize|orientation

今日重点

操作短信数据库　用到contentprovide的知识 使用contentresolver调用具体的方法

操作联系人数据库
提供数据库给别人使用 contentprovider具体写法
内容观察者 contentobserver

多媒体
加载图片
使用canvas
播放音频
播放视频

fragment
动画
android反编译
android的通知