Android移动开发基础案例教程

第8章 ContentProvider（内容提供者）

- 内容提供者简介
- 内容提供者的创建
- 内容提供者的使用
- 内容观察者的使用

学习目标

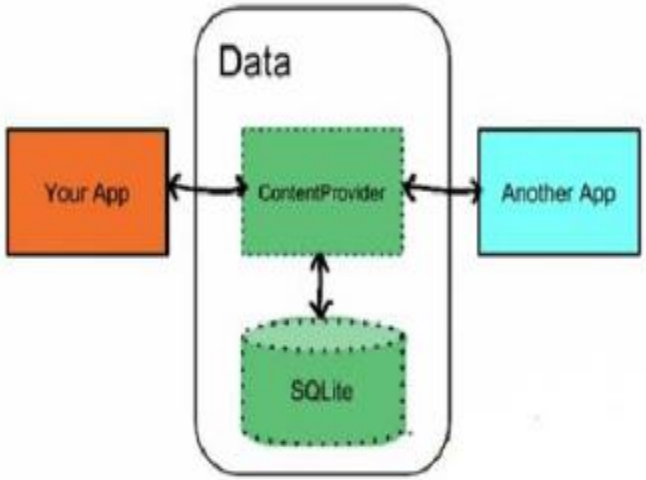
1
内容提供者的创建
内容提供者的使用

2
内容提供者的工作原理

3
内容观察者的使用

ContentPrivoder

- ContentPrivoder是Android平台中的一组件。它是不同应用程序之间数据交换的标准API，主要实现数据在不同应用程序之间的共享，从而能够让其他的应用保存或读取此ContentProvider的各种数据类型。



ContentPrivoder

- 一个程序可以通过实现一个ContentProvider的抽象接口，将自己的数据以Uri形式完全暴露出去。其他应用程序就可以使用ContentResolver，根据Uri访问操作指定数据,对应用中的数据进行添删改查。
- 标准的ContentProvider：Android提供了一些已经在系统中实现的标准ContentProvider，比如联系人信息，图片库等等，可以用这些ContentProvider来访问设备上存储的联系人信息、图片等等。

URI，通一资源标志符(Uniform Resource Identifier， URI)，表示的是web上每一种可用的资源，如 HTML文档、图像、视频片段、程序等都由一个URI进行定位的。

URI的结构组成

URI通常由三部分组成：①访问资源的命名机制；②存放资源的主机名；

③资源自身的名称。

如： https://blog.csdn.net/qq_32595453/article/details/79516787

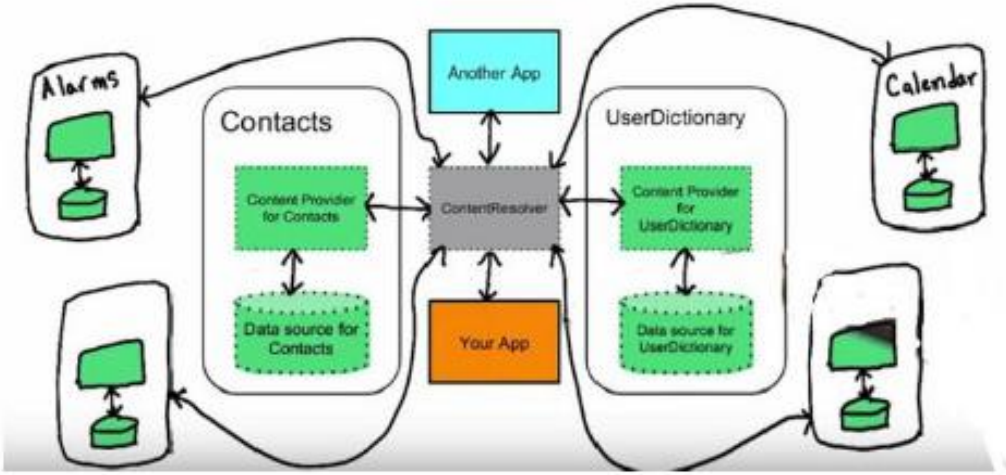
①这是一个可以通过https协议访问的资源，

②位于主机 blog.csdn.net上，

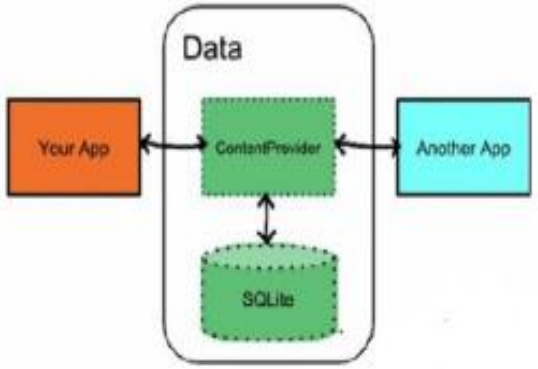
③通过“/qq_32595453/article/details/79516787”可以对该资源进行唯一标识

ContentResolver

- 手机中可不是只有一个Provider内容，它可能安装了很多含有Provider的应用，比如联系人应用，日历应用，字典应用等等。有如此多的Provider，如果开发一款应用要使用其中多个，如果去了解每个ContentProvider的不同实现，是不现实。所以Android为我们提供了ContentResolver来统一管理与不同ContentProvider间的操作。



- 外界的程序通过ContentResolver可以访问ContentProvider提供的数据，Uri是ContentResolver和ContentProvider进行数据交换的标识，根据它区别不同的ContentProvider的。



ContentProvider的基本使用

- 1.创建一个自定义ContentProvider的方式是继承ContentProvider类并实现其六个抽象方法
  - ✓ onCreate()：代表ContentProvider的创建，可以用来进行一些初始化操作
  - ✓ getType(Uri uri)：用来返回一个Uri请求所对应的MIME类型
  - ✓ insert(Uri uri, ContentValues values)：插入数据
  - ✓ query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)：查询数据
  - ✓ update(Uri uri, ContentValues values, String selection, String[] selectionArgs)：更新数据
  - ✓ delete(Uri uri, String selection, String[] selectionArgs)：删除数据

ContentProvider的基本使用

- 2.需要在AdnroidManifest.xml中对ContentProvider进行注册
  - ✓ <provider android:name=".MyContentProvider"
  - ✓ android:authorities="com.li.example.MyContentProvider"
  - ✓ android:exported="true" />
  - ✓ name：ContentProvider的全称类名
  - ✓ authorities：唯一标识了一个ContentProvider，外部应用通过该属性值来访问我们的ContentProvider。因此该属性值必须是唯一的，建议在命名时以包名为前缀
  - ✓ exported：表明是否允许其他应用调用ContentProvider，true表示支持，false表示不支持。默认值根据开发者的属性设置而会有所不同，如果包含 Intent-Filter 则默认值为true，否则为false

ContentRerolver

- 使用Context提供的getContentResolver()获取实例，
- 查询
  - ✓ public final Cursor query(Uri uri, String[] projection,
    String selection, String[] selectionArgs, String sortOrder);
- 新增
  - ✓ public final Uri insert(Uri url, ContentValues values)
- 更新
  - ✓ public final int update(Uri uri, ContentValues values, String where,
    String[] selectionArgs)
- 删除
  - ✓ public final int delete(Uri url, String where, String[] selectionArgs)

URI的操作工具

- ContentProvider中操作的数据可以都是从SQLite数据库中获取的，而数据库中可能存在许多张表，这时候就需要用到Uri来表明是要操作哪个数据库、操作数据库的哪张表了.那么如何确定一个Uri是要执行哪项操作呢？这里需要用到UriMatcher来帮助ContentProvider匹配Uri.

- UriMatcher工具类（两个方法）

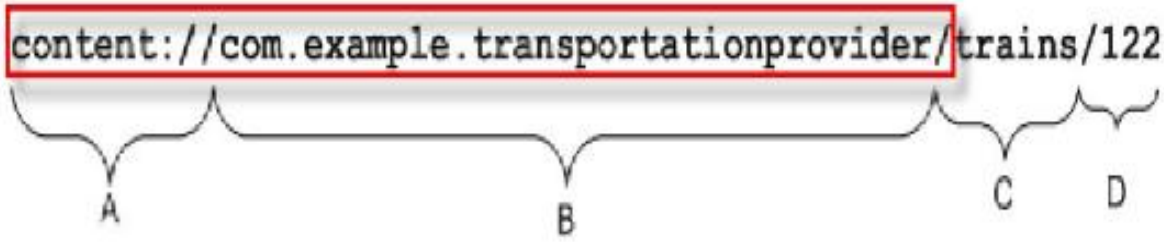
  - ✓ void addURI(String authority, String path, int code)

  authtity是在AndroidManifest.xml中注册的ContentProvider的authority属性值；path表示一个路径，可以设置为通配符，#表示任意数字，*表示任意字符；两者组合成一个Uri，而code则代表该Uri对应的标识码

  - ✓ int match(Uri uri)

  用于判断Uri的标识码是否对应，如果对应则返回标识码的值，找不到对应的Uri标识码则方法返回-1；

对Uri参数有判断的Contentprovider



C部分为Uri的资源部分，D部分为该资源的数据ID，指定了数据项，如果没有指定数据ID，则通常代表访问所有数据项。

ContentUris工具类

- 有两个常用方法：

withAppenddedld(Uri, id):用于路径加上ID部分；

parseid(uri):用于从指定uri中解析粗所包含的ID值。

```
//生成后的Uri为：
content://com.example.testcp.FirstContentProvider/users/1
Uri uri = Uri.parse( “content://com.example.li/users")
Uri insertedUserUri = ContentUris.withAppendedId(uri, 1);
```

```
Uri insertedUserUri =
Uri.parse("content://com.example.testcp.FirstContentProvider/u
sers/1")
long userId = ContentUris.parseId(insertedUserUri);
```

UriMatcher类使用介绍

第一步把你需要匹配Uri路径全部给注册上，如下：

```
//常量UriMatcher.NO_MATCH表示不匹配任何路径的返回码
UriMatcher  sMatcher = new UriMatcher(UriMatcher.NO_MATCH);
//如果match()方法匹配content://com.ljq.provider.personprovider/person路径，返回匹配码为1
sMatcher.addURI("com.li.provider.personprovider", "person", 1);//添加需要匹配uri，如果匹配就会返回匹配码
//如果match()方法匹配content://com.ljq.provider.personprovider/person/230路径，返回匹配码为2
sMatcher.addURI("com.li.provider.personprovider", "person/#", 2);//#号为通配符
switch (sMatcher.match(Uri.parse("content://comli.provider.personprovider/person/10")))
{
  case 1
    break;
  case 2
    break;
  default://不匹配
    break;
}
```

ContentUris类使用介绍

ContentUris类用于操作Uri路径后面的ID部分，它有两个的方法：
withAppendedId(uri, id)用于为路径加上ID部分：

```
Uri uri =
Uri.parse("content://com.li.provider.personprovider/person")
Uri resultUri = ContentUris.withAppendedId(uri, 10);
//生成后的Uri为：
content://com.li.provider.personprovider/person/10
```

parseId(uri)方法用于从路径中获取ID部分：

```
Uri uri =
Uri.parse(“content://com.li.provider.personprovider/person/10”);
long personid = parseid(uri):
```

ContentPrivoder实现步骤

- 开发一个ContentProvider的子类，该子类实现增、删、改、查等方法。
- 注册URI
- 在AndroidManifest.xml文件中注册该ContentProvider
- 利用URI，使用ContentRrsovler访问。

示例，简单的contentProvider

- 新建一工程ContentProviderExam继承contentProvider，实现相关的方法，定义好URI，在每个方法中，只是打印信息。
- 在androidManifest.xml文件中添加相关的说明。
  - ✓代码见ContentProviderTest－mycontentproviderexam应用
- 再建立一个工程，使用这个provider。单击四个按钮，查看日志
  - ✓代码见ContentProviderTest－mycontentpresovlerexam应用

示例，简单的contentProvider

- 1新建一工程
- 2新新ContentProviderExam继承contentProvider，实现相关的方法，定义好URI，在每个方法中，只是打印信息。

示例，简单的contentProvider

```
public class ContentProviderExam extends ContentProvider {
    public ContentProviderExam() {    }
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
       Log.d("contentProvider---->>>","Delete");
       return 0;    }
    @Override
    public String getType(Uri uri) {
        return null;    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
         Log.d("contentProvider---->>>","insert");
        return null;    }

    @Override
    public boolean onCreate() {
         Log.d("contentProvider---->>>","onCreate");
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        Log.d("contentProvider---->>>","query");
        return null;
    }
   @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
          Log.d("contentProvider---->>>","update");
```

示例，简单的contentProvider

- 3在androidManifest.xml文件中添加相关的说明

```
<provider
    android:name=".ContentProviderExam"
    android:authorities="com.li.provider.exam"
    android:enabled="true"
    android:exported="true"></provider>
```

- 4.运行这个工程，可以使用有activity的，也可以使用没有页面的。

示例，简单的contentProvider

- 5 新建工程，mycontentpresovlerexam

在Activity中定义引用字符串
final String **tmp="content://com.li.provider.exam/"**;
定义对象 ContentResolver resolver;

- 5定义四个按钮，完成增删改查
- 6获取ContentResolver

```
resolver=getContentResolver();
//插入操作
ContentValues values=new ContentValues();
values.put("name","dddd");
resolver.insert(Uri.parse(tmp),values);

resolver.query(Uri.parse(tmp),null,null,null,null);

resolver.delete(Uri.parse(tmp),null,null);

ContentValues values=new ContentValues();
values.put("name","dddd");
resolver.update(Uri.parse(tmp),values,null,null);
```

```
D/contentProvider----->>>: onCreate
D/contentProvider----->>>: insert
D/contentProvider----->>>: query
D/contentProvider----->>>: Delete
D/contentProvider----->>>: update
```

复杂的应用－生词本

- 创建一个生词本应用，提供一个MyDictContentprovider，程序自己也可以使用，也提供其他程序使用
- 创建一个应用，使用上面的MyDictContentprovider。

复杂的应用—生词本

1.建立DbHelper

```
public class MyDBHelper extends SQLiteOpenHelper {
    private static final String DATA_BASE_NAME="my.db";
    final String CREATE_TABLE_SQL =
            "create table dict(_id integer primary key autoincrement , word , detail)";

    public MyDBHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, DATA_BASE_NAME, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_TABLE_SQL);
    }

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

    }
}
```

复杂的应用—生词本

- 2.创建Word类，定义常量

```
public final class Words {
    public static final String AUTHORITY="com.li.example.words";
    public static final class Word {
        public static final  String _ID="_id";
        public static final  String WORD="word";
        public static final  String DETAIL="detail";
        public  static final Uri DICT_CONTENT_URI=
                Uri.parse("content://"+AUTHORITY+"/words");
        public static  final  Uri WORD_CONENT_URI=
                Uri.parse("content://"+AUTHORITY+"/word");
    }
}
```

复杂的应用—生词本

- 3.定义类继承ContentProvider,定义相关方法，
- 定义返回码，注册URI，定义数据库对象

```
public class MyContentProvider extends ContentProvider {
        private  static UriMatcher matcher=new UriMatcher(UriMatcher.NO_MATCH);
        private static final  int WORDS=1;
        private static final int WORD=2;

        static {
            matcher.addURI(Words.AUTHORITY,"words",WORDS);
            matcher.addURI(Words.AUTHORITY,"word/#",WORD);
        }
    MyDBHelper dbHelper;
    public MyContentProvider() {
    }
```

- 4.重写onCreate方法，创建数据库对象。

```
    @Override
    public boolean onCreate() {
        // TODO: Implement this to initialize your content provider on startup.
        dbHelper=new MyDBHelper(this.getContext(),null,null,1);
        return true;
    }
```

复杂的应用—生词本

5.重写insert方法

```
@Override
public Uri insert(Uri uri, ContentValues values) {
    // TODO: Implement this to handle requests to insert a new row.
    SQLiteDatabase db=dbHelper.getWritableDatabase();
    long rowId=db.insert("dict",Words.Word._ID,values);
    if(rowId>0){
        Uri wordUri= ContentUris.withAppendedId(uri,rowId);
        getContext().getContentResolver().notifyChange(uri,null);
        return wordUri;
    }
    return null;
}
```

复杂的应用—生词本

6.重写delete方法

```
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
     SQLiteDatabase db = dbHelper.getReadableDatabase();
    // 记录所删除的记录数
    int num = 0;
    // 对于uri进行匹配。
    switch (matcher.match(uri))
    {       // 如果Uri参数代表操作全部数据项
        case WORDS:
            num = db.delete("dict", selection, selectionArgs);
            break;
        // 如果Uri参数代表操作指定数据项
        case WORD:
            // 解析出所需要删除的记录ID
            long id = ContentUris.parseId(uri);
            String whereClause = Words.Word._ID + "=" + id;
            // 如果原来的where子句存在，拼接where子句
            if (selection != null && !selection.equals(""))
            {               whereClause = whereClause + " and " + selection;          }
            num = db.delete("dict", whereClause, selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("未知Uri:" + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null); // 通知数据已经改变
    return num;
}
```

7.重写query方法

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    switch (matcher.match(uri))
    {
        // 如果Uri参数代表操作全部数据项
        case WORDS:
            // 执行查询
            return db.query("dict", projection, selection,
                    selectionArgs, null, null, sortOrder);
        // 如果Uri参数代表操作指定数据项
        case WORD:
            // 解析出想查询的记录ID
            long id = ContentUris.parseId(uri);
            String whereClause = Words.Word._ID + "=" + id;
            // 如果原来的where子句存在，拼接where子句
            if (selection != null && !"".equals(selection))
            {
                whereClause = whereClause + " and " + selection;
            }
            return db.query("dict", projection, whereClause, selectionArgs,
                    null, null, sortOrder);
        default:
            throw new IllegalArgumentException("未知Uri:" + uri);
    }

}
```

8.重写update方法

```
@Override
public int update(Uri uri, ContentValues values, String selection,
                  String[] selectionArgs) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    // 记录所修改的记录数
    int num = 0;
    switch (matcher.match(uri))
    {
        // 如果Uri参数代表操作全部数据项
        case WORDS:
            num = db.update("dict", values, selection, selectionArgs);
            break;
        // 如果Uri参数代表操作指定数据项
        case WORD:
            // 解析出想修改的记录ID
            long id = ContentUris.parseId(uri);
            String whereClause = Words.Word._ID + "=" + id;
            // 如果原来的where子句存在，拼接where子句
            if (selection != null && !selectionArgs.equals(""))
            {
                whereClause = whereClause + " and " + selection;
            }
            num = db.update("dict", values, whereClause, selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("未知Uri:" + uri);
    }
    // 通知数据已经改变
    getContext().getContentResolver().notifyChange(uri, null);
    return num;

}
```

9.重写getType方法

```
@Override
public String getType(Uri uri) {
    switch (matcher.match(uri))
    {
        // 如果操作的数据是多项记录
        case WORDS:
            return "vnd.android.cursor.dir/dict";
        // 如果操作的数据是单项记录
        case WORD:
            return "vnd.android.cursor.item/dict";
        default:
            throw new IllegalArgumentException("未知Uri:" + uri);
    }

}
```

在myDictContenprovider中使用自己的生词本

- 由于是本地的数据操作，可直接操作数据库,类似前面的数据库操作。

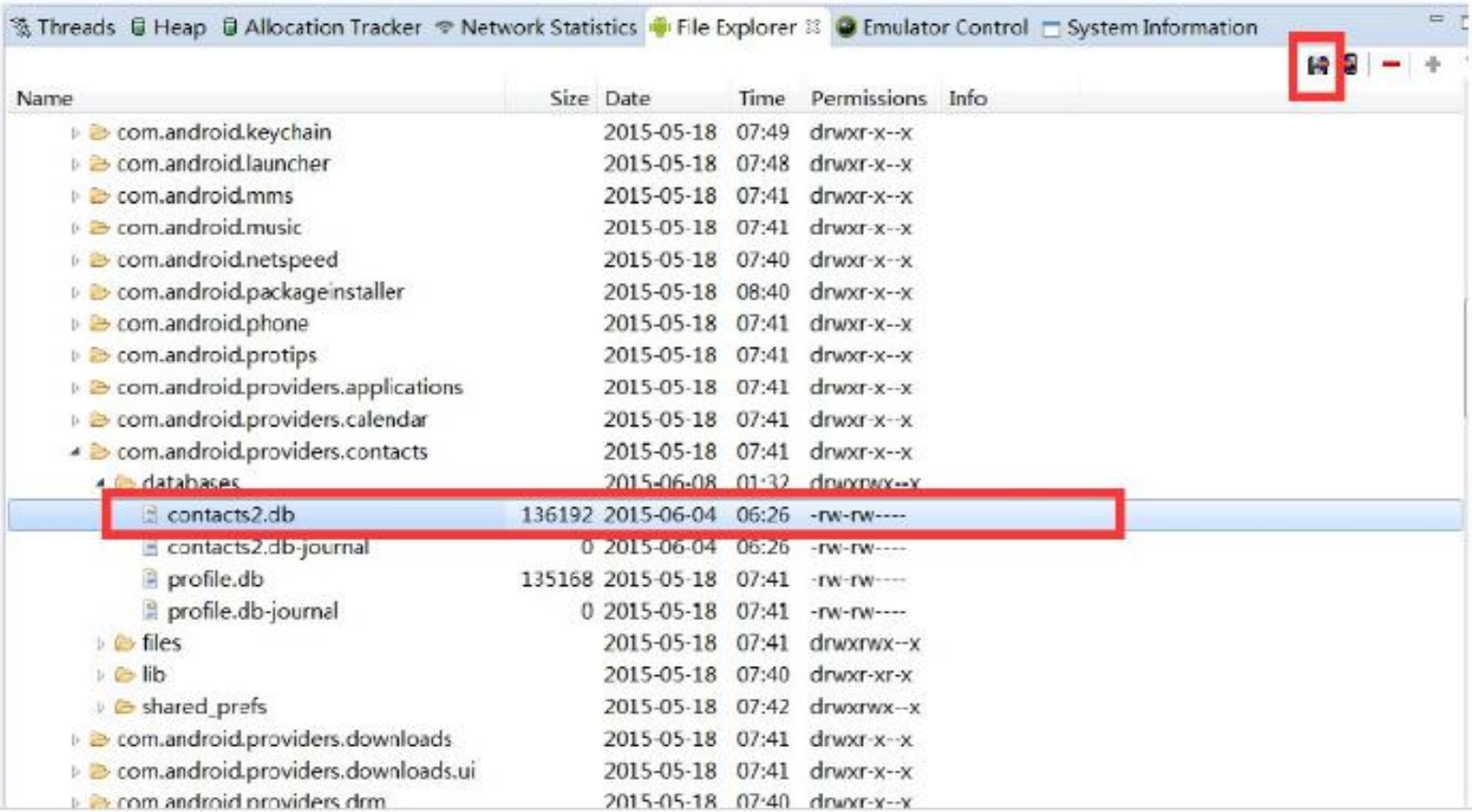
在myDictResovler中使用

- 1.新建工程，将Word类复制过来，保持常量的统一。
- 2.创建三个activity，用来插入，查找，更新和删除。

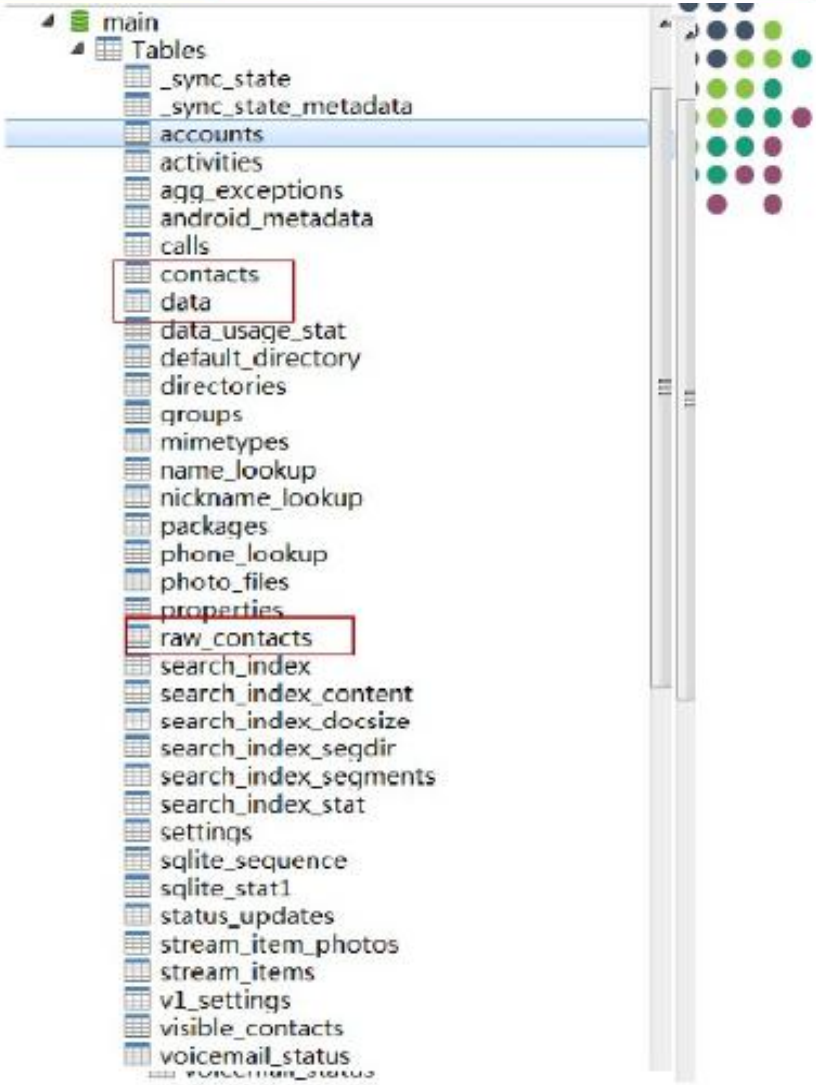
访问系统的ContentProvider—管理联系人

- 代码见ContentProviderTest-ContactReadWrite应用
- 电话本主要信息都存在三张表里
  - ✓ContactsContract.Data
  - ✓ContactsContract.RawContacts
  - ✓ContactsContract.Contacts

导出联系人数据库，打开DDMS



利用工具找到三个表


main
Tables
_sync_state
_sync_state_metadata
accounts
activities
agg_exceptions
android_metadata
calls
contacts
data
data_usage_stat
default_directory
directories
groups
mimetypes
name_lookup
nickname_lookup
packages
phone_lookup
photo_files
properties
raw_contacts
search_index
search_index_content
search_index_docsize
search_index_segdir
search_index_segments
search_index_stat
settings
sqlite_sequence
sqlite_stat1
status_updates
stream_item_photos
stream_items
v1_settings
visible_contacts
voicemail_status

Drag a column header here to group by that column

	_id	display_name	photo_id	custom_ringtone	send_to_voicemail	times_contacted	last_time_contacted	starred	in_visible_group	has_phone_number	lookup	status_update_id	single_is_restricted
1	4	张三			0	0	0	0	1	1	0nE027180418		0
2	5	李四			0	0	0	0	1	1	0nE037D41628		0

Drag a column header here to group by that column

	_id	package_id	mimetype_id	raw_contact_id	is_primary	is_super_primary	data_version	data1	data2	data3	data4	data5	data6	data7	data8	data9	data10	data11	data12	data13	data
1	11		6	4	0	0	0	1-234-56	2		654321										
2	12		4	4	0	0	0	张三	张三												
3	13		6	5	0	0	0	654-321	1		123456										
4	14		4	5	0	0	2	李四	李四												
5	15		6	5	0	0	0	987-654-321	2		123456789										
6	16		2	5	0	0	0	123456	3			4									
7	17		1	5	0	0	0	wssqj@126.com	1												

Drag a column header here to group by that column

	_id	mimetype
1	1	vnd.android.cursor.item/email_v2
2	2	vnd.android.cursor.item/im
3	3	vnd.android.cursor.item/postal-address_v2
4	4	vnd.android.cursor.item/name
5	5	vnd.android.cursor.item/photo
6	6	vnd.android.cursor.item/phone_v2
7	7	vnd.android.cursor.item/group_membership

Contacts表

- contacts表主要存储联系人lookup（查找联系人的关键）
- contact表的数据是由系统组合raw_contact表中的数据而自动生成。contact表中的一行表示一个联系人。
- 主要字段：
  - _id：主键，表的ID，系统自动生成，是联系人的唯一标识
  - lookup：存储查找联系人的lookup信息
  - has_number_phone：联系人电话个数（0表示联系人无电话）
  - photo_id：头像的ID
  - times_contacted：通话记录的次数

表明：我们不可以直接向contacts表中插入数据

raw_contacts表

- 主要存储了联系人的名称、名称的字母索引和帐户类型信息（区别是本机号码还是SIM卡号码）
- 主要字段：
  - _id：主键，表的ID
  - display_name: 联系人名称
  - phonebook_label：名称的字母索引
  - account_id：帐户类型
  - contact_id：外键，与contacts表的_ID字段关联(这个id值是系统生成的)

data表

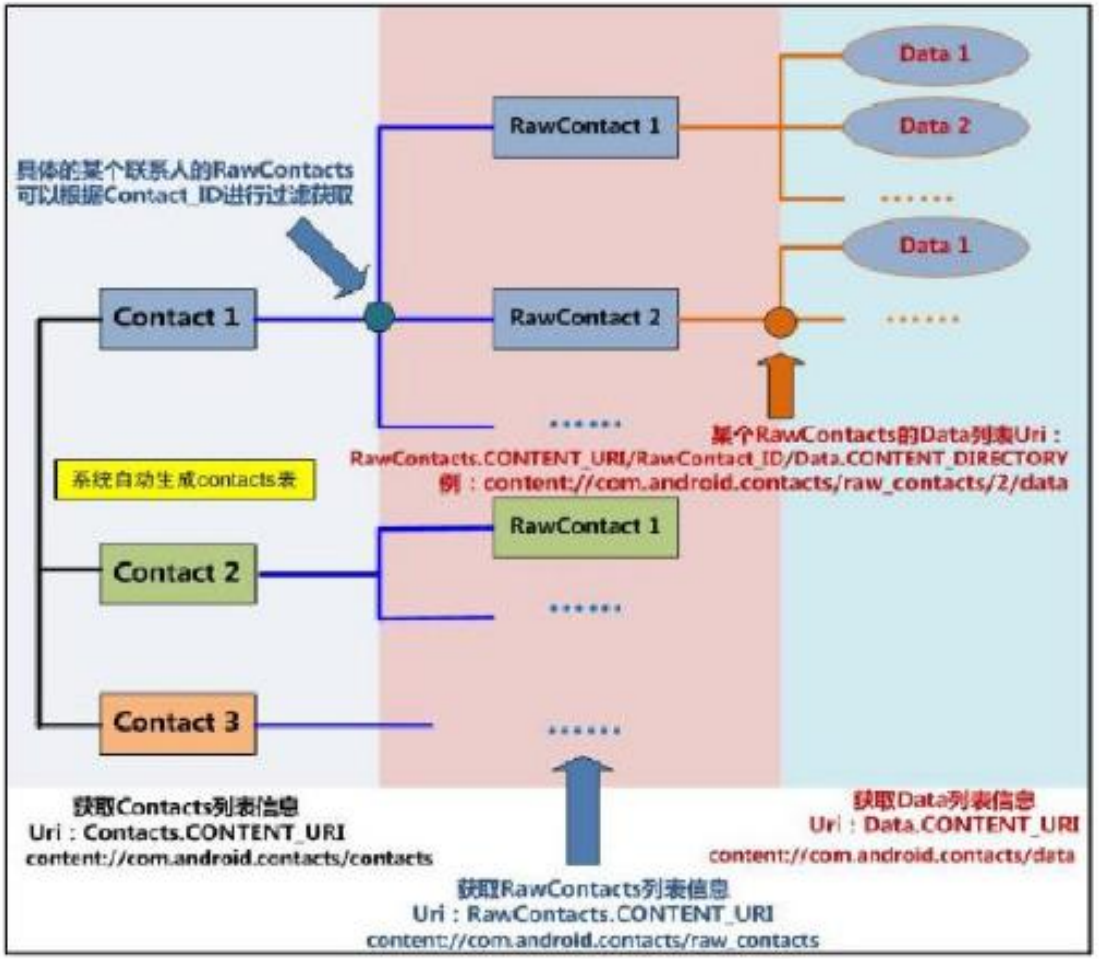
- raw_contact_id：
  外键，与raw_contact表中相对的数据。
- mimetype_id:存储数据的mime类型，与mimetype表相对。如1对应email,7联系人姓名
- _id：主键，表的ID
- data1 到 data15：

一般data1：存放主信息（如联系人名称、电话、Email信息）

data2：存放data1信息对应的类型，如1(Phone.TYPE_HOME)表示家庭座机，2(Phone.TYPE_MOBILE)表示为手机

三个表关系

三张表关系


Data 1
Data 2
RawContact 1
具体的某个联系人的RawContacts
可以根据Contact_ID进行过滤获取
Data 1
Contact 1
RawContact 2
某个RawContacts的Data列表Uri：
RawContacts.CONTENT_URI/RawContact_ID/Data.CONTENT_DIRECTORY
例：content://com.android.contacts/raw_contacts/2/data
系统自动生成contacts表
RawContact 1
Contact 2
Contact 3
获取Contacts列表信息
Uri：Contacts.CONTENT_URI
content://com.android.contacts/contacts
获取Data列表信息
Uri：Data.CONTENT_URI
content://com.android.contacts/data
获取RawContacts列表信息
Uri：RawContacts.CONTENT_URI
content://com.android.contacts/raw_contacts

申请权限

静态申请

在AndroidManifest.xml中进行申请。<manifest ...>

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

动态申请

版本大于等于6.0且当前页面没有这个权限。

主要使用**requestPermissions(permission名字的数组,整型的自定义reques code)**

流程为：发起申请和申请结果

一些准备工作和URI

（1）添加读、写联系人的权限

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

（2）联系人信息的URI：

- ContactsContract.Contacts.CONTENT_URI
- 或content://com.android.contacts/contacts

可以视为全局的URI：
涉及contacts+raw_contacts+data三张表的信息

（3）raw_contacts表信息的URI：

raw_contacts表的URI

- ContactsContract.RawContacts.CONTENT_URI
- 或content://com.android.contacts/rawcontacts

（4）data表信息的URI：

data表的URI

- ContactsContract.Data.CONTENT_URI
- 或content://com.android.contacts/data

（5）联系人电话的URI：

联系人的电话信息URI（系统为方便查找而新新添）

- ContactsContract.CommonDataKinds.Phone.CONTENT_URI
- 或content://com.android.contacts/data/phones

访问系统的ContentProvider—管理联系人

```
//查询所有的联系人及电话，得到了两个list，之后可以将list在listview上显示
final ArrayList<String> names=new ArrayList<>();
final ArrayList<ArrayList<String>> details=new ArrayList<ArrayList<String>>();
resolver=getContentResolver();
Cursor cursor=resolver.query(ContactsContract.Contacts.CONTENT_URI,null,null,null,null);
while(cursor.moveToNext()){
    String _id=cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts._ID));
    String name=cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
    names.add(name);
    Cursor cursor1=resolver.query(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,null,
            ContactsContract.CommonDataKinds.Phone.CONTACT_ID+"="+_id,null,null);
    ArrayList<String>detail=new ArrayList<>();
    while(cursor1.moveToNext()){
        String
num=cursor1.getString(cursor1.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER));
        detail.add(num);
    }
    cursor1.close();
    details.add(detail);
}
cursor.close();
```

访问系统的ContentProvider—管理联系人

```
//添加联系人的姓名和电话
String nam=name.getText().toString();
 String phon=phone.getText().toString();
 String emai=email.getText().toString();
 ContentValues values=new ContentValues();
Uri uri= resolver.insert(ContactsContract.RawContacts.CONTENT_URI,values);
 long _id= ContentUris.parseId(uri);
 values.clear();
 values.put(ContactsContract.Data.RAW_CONTACT_ID,_id);
 values.put(ContactsContract.Data.MIMETYPE,
ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE);
 values.put(ContactsContract.CommonDataKinds.StructuredName.GIVEN_NAME,nam);
 resolver.insert(ContactsContract.Data.CONTENT_URI,values);
 values.clear();
 values.put(ContactsContract.Data.RAW_CONTACT_ID,_id);
 values.put(ContactsContract.Contacts.Data.MIMETYPE,
ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE);
 values.put(ContactsContract.CommonDataKinds.Phone.TYPE,
ContactsContract.CommonDataKinds.Phone.TYPE_MOBILE);
 values.put(ContactsContract.CommonDataKinds.Phone.NUMBER,phon);
 resolver.insert(ContactsContract.Data.CONTENT_URI,values);
```

管理多媒体内容

- 音乐URI(表)
  MediaStore.Audio.Media.EXTERNAL_CONTENT_URI=content://media/external/audio/media
- 专辑URI(表)
  MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI=//content://media/external/audio/albums
- 图片URI(表)
  MediaStore.Images.Media.EXTERNAL_CONTENT_URI=content://media/external/images/media
- 视频URI(表)
  MediaStore.Video.Media.EXTERNAL_CONTENT_URI=content://media/external/video/media

文档

管理多媒体内容

添加权限

- ✓ <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
- ✓ <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

示例一显示SD卡上的所有图片

- ContentProviderTest-readPicture应用。

监听ContentProvider的数据改变

- ContentProvider将数据共享出来，ContentResovler会根据业务需要去主动查询共享数据。有时，应用程序需要监听contentProvider所共享数据的改变，并随着数据改变而提供响应。这是就会用到ContentObserver
- 使用ContentProvider时，不管实现insert，delete或是update方法中的哪一个，只要数据改变，程序就会调用.
  - ✓getContext().getContentResolver().notifyChange(uri,null);

ContentObserver

- 也称内容观察者，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。
- 注册/取消注册ContentObserver方法，抽象类ContentResolver类中的方法原型如下：
  - ✓ public final void **registerContentObserver**(Uri uri, boolean notifyForDescendents, ContentObserver observer)
  - ✓ 功能：为指定的Uri注册一个ContentObserver派生类实例，当给定的Uri发生改变时，回调该实例对象去处理。

ContentObserver

- 参数：**uri** 需要观察的Uri(需要在UriMatcher里注册，否则该Uri也没有意义了)
- **notifyForDescendents** 为false 表示精确匹配，即只匹配该Uri为true 表示可以同时匹配其派生的Uri，举例如下：
- 假设UriMatcher 里注册的Uri共有一下类型：
  - ✓ content://com.qin.cb/student (学生)
  - ✓ content://com.qin.cb/student/#
  - ✓ content://com.qin.cb/student/schoolchild(小学生，派生的Uri)
- 我们当前需要观察的Uri为content://com.qin.cb/student，如果发生数据变化的 Uri 为 content://com.qin.cb/student/schoolchild，当notifyForDescendents为 false，那么该ContentObserver会监听不到，当notifyForDescendents 为ture，能捕捉该Uri的数据库变化。

ContentObserver

- public final void **unregisterContentObserver**(ContentObserver observer)
- 功能：取消对给定Uri的观察
- 参数： observer ContentObserver的派生类实例

ContentObserver类介绍

- public void ContentObserver(Handler handler)
  - ✓ 说明：所有 ContentObserver的派生类都需要调用该构造方法
- void **onChange**(boolean selfChange)
  - ✓ 功能：当观察到的Uri发生变化时，回调该方法去处理。所有ContentObserver的派生类都需要重载该方法去处理逻辑。
- void **onChange**(boolean selfChange， Uri uri)
  - ✓ 功能：当观察到的Uri发生变化时，回调该方法去处理。

实例，监听接收到的短信

- Sms的URI
  - ✓content://sms/inbox　　收件箱
  - content://sms/sent　　已发送
  - content://sms/draft　　草稿
  - content://sms/outbox　　发件箱
  - content://sms/failed　　发送失败
  - content://sms/queued　　待发送列表

8.4.2 实战演练——监测数据的喵

1 功能描述： 监测数据库的变化。

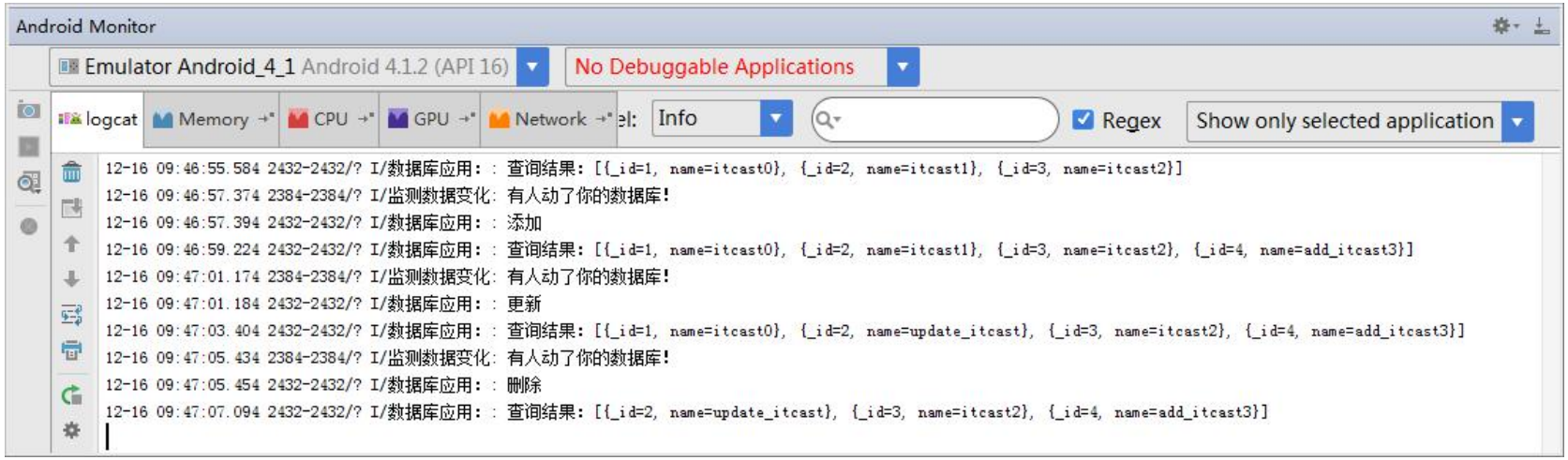
2 技术要点： 内容观察者的工作原理及用法。

3 实现步骤：

① 创建操作数据库的程序
② 用户交互界面的设计与实现
③ 数据库的帮助类（PersonDBOpenHelper.java）的创建
④ 内容提供者（PersonProvider.java）的创建
⑤ 操作数据库界面逻辑代码的设计与实现
⑥ 创建监测数据库变化的程序
⑦ 监测数据库变化界面逻辑代码的设计与实现

案例代码（详见教材P12—P22）

8.4.2 实战演练——监测数据的喵



8.5 本章小结

本章详细地讲解了内容提供者的相关知识，首先简单地介绍了内容提供者，然后讲解了如何创建内容提供者以及如何使用内容提供者访问其他程序暴露的数据。最后讲解内容观察者，通过内容观察者观察数据的变化。

至此，Android的四大组件已经讲完了，分别是Activity、Service、BroadcastReceiver和本章所讲的ContentProvider，熟练掌握四大组件的使用有助于初学者们更好的开发程序，因此要求一定要熟练掌握这些组件的使用。

Thank You!