内容提供者ContentProvider

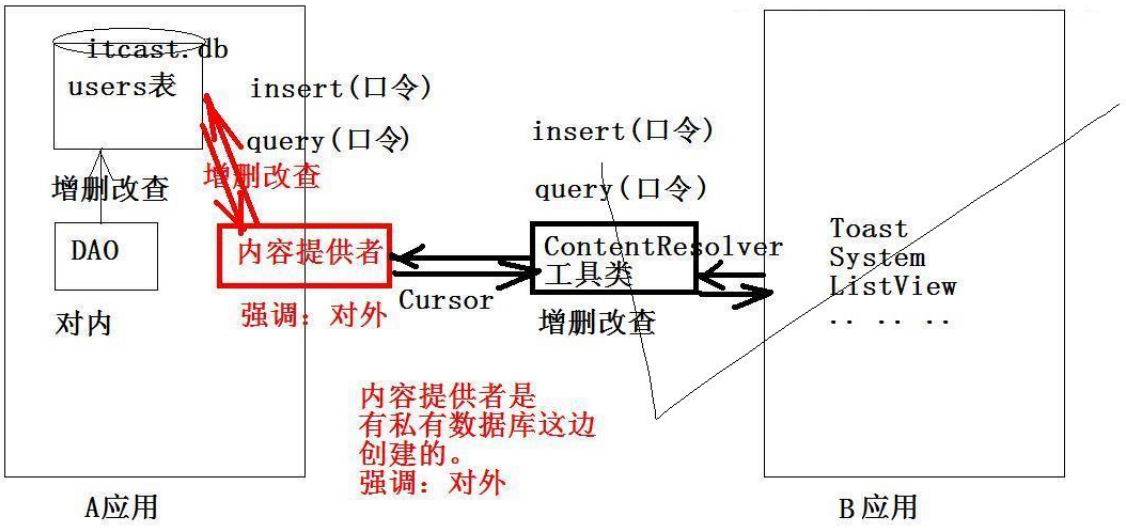
主要用于一个应用访问别一个应用私有的数据库进行数据的增删改查等操作

当你想让自愿暴露某个应用下私有的数据库，此时该应用就可以使用内容提供者这个技术

为不同的软件之间数据共享，提供统了一套接口。也就是说，如果我们想让其他的应用使用我们自己程序内的数据，就可以使用ContentProvider定义一组对外开放的接口，从而使得其他的应用可以使用咱们应用的文件、数据库内存储的信息。

原理： 通过口令机制来判断是否可以访问

内容提供者定义了一组对外开放的接口，使其他应用可以访问自己的应用的数据库内容，普通外部应用不可以直接访问私有的数据库，只能通过内容提供者访问私有数据库，内容提供者处于应用内部，在内容提供者中定义了一些访问路径匹配等操作，当外部应用通过路径访问私有数据库时，内容提供者根据路径匹配出具体的操作，将私有数据返回给外部应用。




## 内容提供者中口令三大部份：

**前缀(必写固定): // 权限(必写) / 路径数据库表名(可选)**

**content: // cn.itheima.provider.itcas / users**

## 访问已有的数据库

通过**SQLiteDatabase.openDatabase**打开已有的数据库增册改查

参一：需要打开数据库文件的路径

参二：游标工厂

参三：标识符，只读打开，读写打开

SQLiteDatabase.openDatabase("数据库文件路径", null, SQLiteDatabase.OPEN_READONLY);

然后通过增删改查对数据库操作。

## 第三方私有数据库访问方式：

一、通过命令改变私有数据库的权限

adb shell------>cd 应用程序包名目录-------->cd 私有数据库目录--------->chmod 777 数据库名.db

二、是通过第三方应用提供的对外接口访问

1、创建工具类对象 ContentResolver cr = this.getContentResolver();

2、准备访问内容提供者的口令，大小写必须一致

参一说明：就是口令三大部份

第一部分：固定 content://

第二部分：去内容提供者的清单文件中找注册authorities

第三部分：去内容提供者的类里去找搜索UriMatcher找到.addURI方法中的第二个参数就是

Uri uri = Uri.parse("content://cn.itheima.provider.itcast/users");

3、通过对应的增删改查命令访问数据库。正常数据库操作即可。

增：

```
ContentValues values = new ContentValues();
values.put("name", "zhaoqi");
values.put("money", "1000");     <=====>（列名字段，值）
cr.insert(uri, values);
```

(用这个方法可以省略第一步创建工具对象)getContentResolver().insert(uri, values)

删：

```
getContentResolver().delete(uri, "name=?", new  String[]{"zhaoqi"});
```

改：

参一：uri口令

参二：values对象的数据

参三：修改条件

参四：  条件值

```
ContentValues values = new ContentValues();
values.put("money", "20000"); <=====>（列名字段，值）
getContentResolver().update(uri, values, "name=?",  new String[]{"zhaoqi"});
```

查：

查询方法参数：1、uri

2、表中的字段  new  String[]{"表中列名1","表中列名2"...}

3、以什么条件查 "字段=？"

4、条件值  new  String[]{ 条件字段值 }

5、排序方式  一般为null

```
Cursor cursor = getContentResolver().query(uri, null, null, null,  null);
//查表中的所有数据
                if (cursor!=null && cursor.getCount()>0) {
                    while (cursor.moveToNext()) {
    String name = cursor.getString(cursor.getColumnIndex("name"))
    String money =cursor.getString(cursor.getColumnIndex("money"));
            System.out.println("name"+name + "money"+money);      }
```

## 创建内容提供者对外提供访问私有数据库

1、创建数据库助手类  继承extends SQLiteOpenHelper

2、创建一个类继承ContentProvider，实现其中的增删改查方法,及onCreate()方法

onCreate()/query()/insert()/delete()/update()方法

onCreate()方法在创建内空提供者对象时调用可以在这个方法中创建数据库助手对象

3、注册内容提供者

name表示内容提供者全路径名

authorities表示访问内容提供者中URI口令的第二个部份　可以是任意字符串

exported表示是否自愿将内容暴露给第三方，true自愿

```
<provider
        android:name="cn.itheima.provider.UserProvider"      内容提供者类
        android:authorities="cn.itheima.provider.itcast"        口令第二部分
        android:exported="true" >
</provider>
```

4、在类中定义URI正确的口令　最好用静态加载口令

创建URI口令对象：UriMatcher u = new UriMatcher(口令值);　　参数： 口令如果出错了，返回的值

参一：口令的第二部份，和清单文件中配置的相同

参二：口令的第三部份

参三：口令如果正确了，返回的值，int类型

加载口令：　　u.addURI("cn.itheima.provider.itcast" ， "users" ， 口令值);

5、在对应的方法中进行口令的判断

参数一：URI，表示第三方传入的口令　　int code = u.match(uri);　　if(code==口令值 ) { }else{ }

如果口令正确就执行相应的操作，并返回结果。

```
14 public class UserProvider extends ContentProvider {
15     //创建UriMatcher对象
16     //参数一：  口令如果出错了，返回的值
17     private static UriMatcher uriMatcher = new UriMatcher(0);
18     static{
19         //参数一：口令的第二部份，和清单文件中配置的相同
20         //参数二：口令的第三部份
21         //参数三：口令如果正确了，返回的值，int类型
22         uriMatcher.addURI("cn.itheima.provider.itcast","users",1);
23     }
24     private UserOpenHelper userOpenHelper;
25     /**
26      * 内容提供者创建时调用该方法          数据库助手对象
27      */
28     @Override
29     public boolean onCreate() {
30         userOpenHelper = new UserOpenHelper(this.getContext());
31         return false;
32     }
33     @Override
34     public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
       {
35         //判断口令是否正确
36         int code = uriMatcher.match(uri);   //参数一：URI，表示第三方传入的口令
37         //如果口令正常
38         if(code==1){
```

```
            SQLiteDatabase db = userOpenHelper.getReadableDatabase();
            //查询，返回结果有4条记录
            Cursor c = db.query("users", projection, selection, selectionArgs, null, null, sortOrder);
            //返回Cursor
            return c;
        }else{
            //抛出异常
            throw new RuntimeException("口令出错，请重试");
        }
    }
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        //判断口令是否正确
        int code = uriMatcher.match(uri);
        //如果口令正确
        if(code==1){
            SQLiteDatabase db = userOpenHelper.getWritableDatabase();
            db.insert("users",null,values);
            return uri;
        }else{
            throw new RuntimeException("口令出错，请重试");
```

```
<!-- 注册内容提供者
name表示内容提供者全路径名
authorities表示访问内容提供者中URI口令的第二个部份
exported表示是否自愿将内容暴露给第三方，true自愿                    -->
<provider
android:name="cn.itheima.provider.UserProvider"
android:authorities="cn.itheima.provider.itcast"
android:exported="true">
</provider>
```

清单注册内容提供者

```
public class UserOpenHelper extends SQLiteOpenHelper {
    public UserOpenHelper(Context context) {
        //创建数据库文件
        super(context,"itcast.db",null,1);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        //创建数据库文件中的表
        String sql0 = "create table users(id integer primary key autoincrement,name varchar not null,sex varchar
        not null)";
        String sql1 = "insert into users(name,sex) values('哈哈','男')";
        String sql2 = "insert into users(name,sex) values('呵呵','男')";
        String sql3 = "insert into users(name,sex) values('嘻嘻','女')";
        String sql4 = "insert into users(name,sex) values('嘿嘿','男')";
        //执行SQL语句
        db.execSQL(sql0);
        db.execSQL(sql1);
        db.execSQL(sql2);
        db.execSQL(sql3);
        db.execSQL(sql4);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

数据库助手类创建数据库并添加数据

## 访问第三方数据库获取短信，联系人数据

Uri uri = Uri.parse("content://sms");

address：电话号码

date：短信时间

body：短信内容

type：短信类型 2为发送 1为接收

数据库 mmssms

表 sms

字段 body/type/address/date

口令 content://sms

```
<!-- 读系统内置短信需要权限 -->
<uses-permission android:name="android.permission.READ_SMS"/>

    <!-- 写SD卡 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

手机系统联系人和联系人数据库

raw_contacts表

data表

mimetypes表

mimetypes表，id为5表示手机号码，6表示姓名，1表示邮箱。

- raw_contacts表：可以得到所有联系人的id

  contact_id：联系人id

- data表：联系人的具体信息，一个信息占一行

  data1：信息的具体内容

  raw_contact_id：联系人id，描述信息属于哪个联系人

  mimetype_id：描述信息是属于什么类型

mimetypes表：通过mimetype_id到该表查看具体类型

查询联系人的步骤：首先要建一个JAVABean类来获取联系人信息，同时要有一个集合在保存

1、通过raw_contacts获取联系人id，表查询一共有多少个联系人；

2、 cursor.getCount()方法判断表中的联系人是否为空

3、通过联系人id获取data表中的data1字段和mimetypes字段；

4、通过mimetypes表查询类型。

5、 拿到联系人信息后，判断mimetype的类型

代码实现：

1、Uri uri = Uri.parse("content://com.android.contacts/raw_contacts");

Cursor cursor = getContentResolver().query( uri, newString[]{ "contact_id" }, null, null, null );

2、if(cursor.getCount() >0){

cursor.getCount()方法判读有没有联系人，如果没有就不用查寻获取了

while (cursor.moveToNext()) {

获得每一个联系人的ID号

```
String  contact_id  =  cursor.getString(0);
```

查data表最好声明为成员变量

```
Uri  datauri  =  Uri.parse("content://com.android.contacts/data");
```

3、开始查询"data1", "mimetype" 的列名字段以每个人的ID号为查询条件

```
Cursor cursors = getContentResolver().query(datauri,new String[] { "data1", "mimetype" }, "raw_contact_id=?",
new String[] { contact_id }, null);
```

开始循环获取值

```
while  (cursors.moveToNext())  {
   String  data =cursors.getString(0);
4、String  mimetype =cursors.getString(1);
```

开始对mimetype的数据判断是什么

```
5、if  ("vnd.android.cursor.item/phone_v2".equals(mimetype_id))  {
```

```
                mNna.setPhone(data);    数据添加进JAVAean变量
        }
    }
```

得到了如何查询到联系人后，接下来使用ContentResolver来查询联系人信息，那么问题来了，如何知道访问的Uri呢？这又需要通过查看联系人源码才能得到Uri。

vnd.android.cursor.item/photo　　联系人图象

vnd.android.cursor.item/phone_v2　　联系人电话号

vnd.android.cursor.item/name　　联系人姓名

vnd.android.cursor.item/email_v2　　邮箱

vnd.android.cursor.item/postal-address_v2　　通信地址

Data中的常量

Data._ID:　　"_id"

Data.DISPLAY_NAME:　　“display_name”

Data.DATA1:　　“data1”

Data.DATA2:　　“data2”

Data.RAW_CONTACT_ID:　　“raw_contact_id”

Data.MIMETYPE:　　“mimetype”

# 内容观察者ContentResolver

## 主要作用是监听第三方应用私有数据库的变化并通知给用户

利用**ContentResolver**可以获取内容提供者提供的其他应用是私有数据库信息，但是如果有这样的需求，当其他应用的私有数据库发生改变时，我们的应用能够收到数据变化的通知，这里就用到了**ContentObserver**内容观察者来实现