Android中的联系人

Android中的联系人存储是通过ContentProvider实现的。因此APP对系统通讯录进行操作涉及到ContentProvider接口的使用。

# 通讯录存储常用的数据库表

使用有关接口前，首先了解一下通讯录数据库中常用的数据库表：

|  |  |
| --- | --- |
| 表名 | 表用途 |
| contacts | 联系人表，存储了实际的联系人姓名，头像，最后通话时间等信息。 会对实际的联系人数据进行一定去重。 |
| raw\_contacts | 实际的联系人数据表，每一行是一个单独的联系人。 会存在多行对应同一个contacts表中条目的情况。 |
| data | 所有联系人信息数据。通过raw\_contact\_id外键与raw\_contacts建立联系。 |

文档中的描述：

contacts与raw\_contacts的区分

一个raw\_contacts对应一个联系人，程序中或用户操作生成新的联系人，就是直接在这个表中插入新条目。

contacts是实际通讯录中显示的联系人——当raw\_contacts中存在相同名称的联系人时，系统会将这几个联系人合并。

（例如通过通讯录添加两个名字相同的名片，这时系统会提示是否要对这两个名片进行合并。）

data表

1.data表每一行都是一项数据(姓名，电话，Email，网址，生日等)。并通过外键raw\_contacts\_id与raw\_contacts表关联起来。

2.由1所述，一个联系人根据情况会有多条data数据。数据存储在data1-15这15列中。

例如某一行存储电话号码，那么在表中data1列存储电话号码，data2列存储号码类型（单位／家庭／组织等）。

又例如某一行存储的联系人姓名，那么data1列存储显示在界面上的名称，data2存储名，data3存储姓。

3.依数据类型不同，data1-14的含义会不同；data15默认存储blob二进制形式的数据。

4.那么又如何区分不同行数据的真实类型呢？是通过data表中mimetype\_id列的值（整形）来进行区分。根据这一列的取值，对data1-14进行不同的解析。mimetype\_id中数值与类型的对应关系在mimetypes表中定义。例如：

\_id mimetypes 含义

1 vnd.android.cursor.item/email\_v2 电子邮件

2 vnd.android.cursor.item/im 即时通讯

3 vnd.android.cursor.item/nickname 昵称

在编写代码时，实际传入的是mimetypes中的字符串参数，而不是ID值。

以上数据库中所有表及字段的定义，都可在android.provider.ContactsContract中找到。

通讯录存储的数据文件在/data/data/com.android.providers.contacts/databases/目录下，需要手机获取Root权限。

用java的思想理解就是：

假定表contacts 定义为user表，表raw\_contacts定义为user\_detail，

则

* user对user\_detail是一对多关系
* 两个表中用户名称是唯一约束，
* data表是 user\_detail的从表，data表通过外键raw\_contacts\_id与user——detail（raw\_contacts）表关联起来
* 一个人根据情况会有多条data数据。数据存储在data1-15这15列中，
* 注意：data中的16个列数据类型不同，依数据类型不同，data1-14的含义会不同；data15默认存储blob二进制形式的数据。
* 区分不同行数据的真实类型呢？是通过data表中mimetype\_id列的值（整形）来进行区分。根据这一列的取值，对data1-14进行不同的解析。mimetype\_id中数值与类型的对应关系在mimetypes表中定义

附：类型对照表：

\_id mimetypes 含义

1 vnd.android.cursor.item/email\_v2 电子邮件

2 vnd.android.cursor.item/im 即时通讯

3 vnd.android.cursor.item/nickname 昵称

在编写代码时，实际传入的是mimetypes中的字符串参数，而不是ID值。

以上数据库中所有表及字段的定义，都可在android.provider.ContactsContract中找到。

通讯录存储的数据文件在/data/data/com.android.providers.contacts/databases/目录下，需要手机获取Root权限。

# 对通讯录进行增删改查

## select 按电话号码查询联系人

Uri phoneUri = Uri.withAppendedPath(ContactsContract.CommonDataKinds.Phone.CONTENT\_FILTER\_URI, Uri.encode(phone));

ContentResolver resolver = context.getContentResolver();

Cursor cursor = resolver.query(phoneUri, new String[]{ContactsContract.CommonDataKinds.Phone.\_ID,

ContactsContract.CommonDataKinds.Phone.DISPLAY\_NAME, ContactsContract.CommonDataKinds.Phone.CONTACT\_ID}, null, null, null);

while (cursor.moveToNext()) {

int id = cursor.getInt(0);

String name = cursor.getString(1);

int contactId = cursor.getInt(2);

if (name.equals(user.getName())) {

deleteList.add(id);

}

}

注意这里使用的URL是ContactsContract.CommonDataKinds.Phone.CONTENT\_FILTER\_URI，而不是ContactsContract.PhoneLookup.CONTENT\_FILTER\_URI。

这是由于PhoneLookup.CONTENT\_FILTER\_URI会以用户提供的手机号查询后，再使用标准格式的电话号码再次查找，会返回两个相同的结果。例如用户提供了号码17000000000，那么程序会先查询17000000000号码，再查询+86 17000000000，并且两次查询都会成功。

## selectAll查询通讯录中所有联系人

Uri uri = ContactsContract.Data.CONTENT\_URI;

ContentResolver resolver = context.getContentResolver();

Cursor cursorUser = resolver.query(uri, new String[]{ContactsContract.CommonDataKinds.Phone.\_ID,

ContactsContract.CommonDataKinds.Phone.DISPLAY\_NAME, ContactsContract.CommonDataKinds.Phone.RAW\_CONTACT\_ID}, null, null, null);

while( cursorUser.moveToNext()) {

int id = cursorUser.getInt(0); // 按上面数组的声明顺序获取

String name = cursorUser.getString(1);

int rawContactsId = cursorUser.getInt(2);

}

## delete删除联系人某项数据(Data中某一项)

int id; // data表中对应的id值

ArrayList<ContentProviderOperation> ops = new ArrayList<>();

ops.add(ContentProviderOperation.newDelete(ContactsContract.Data.CONTENT\_URI)

.withSelection(ContactsContract.Data.\_ID + "=?", new String[]{String.valueOf(d)})

.build());

context.getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);

## insert向通讯录中添加新的联系人

ArrayList<ContentProviderOperation> ops = new ArrayList<>();

ops.add(ContentProviderOperation.newInsert(ContactsContract.RawContacts.CONTENT\_URI)

.withValue(ContactsContract.RawContacts.ACCOUNT\_NAME, null) // 此处传入null添加一个raw\_contact空数据

.build());

ops.add(ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT\_URI)

.withValueBackReference(ContactsContract.Data.RAW\_CONTACT\_ID, 0) // RAW\_CONTACT\_ID是第一个事务添加得到的，因此这里传入0，applyBatch返回的ContentProviderResult[]数组中第一项

.withValue(ContactsContract.Data.MIMETYPE, ContactsContract.CommonDataKinds.StructuredName.CONTENT\_ITEM\_TYPE)

.withValue(ContactsContract.CommonDataKinds.StructuredName.DISPLAY\_NAME, userName)

.build());

ops.add(ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT\_URI)

.withValueBackReference(ContactsContract.Data.RAW\_CONTACT\_ID, 0)

.withValue(ContactsContract.Data.MIMETYPE, ContactsContract.CommonDataKinds.Phone.CONTENT\_ITEM\_TYPE)

.withValue(ContactsContract.CommonDataKinds.Phone.NUMBER, phoneNumber)

.withValue(ContactsContract.CommonDataKinds.Phone.TYPE, ContactsContract.CommonDataKinds.Phone.TYPE\_WORK)

.build());

context.getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);

其中withValueBackReference接口传参代表此键值是事务中之前操作得到的结果，因此需要传入之前事务的index值。由于添加联系人是在第一步操作，对应结果数组的第0项。

## update向已有联系人中添加新数据

ArrayList<ContentProviderOperation> ops = new ArrayList<>();

ops.add(ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT\_URI)

.withValue(ContactsContract.Data.RAW\_CONTACT\_ID, rawContactsId) // 这里关键是传入正确的raw\_contacts\_id值

.withValue(ContactsContract.Data.MIMETYPE, ContactsContract.CommonDataKinds.Phone.CONTENT\_ITEM\_TYPE)

.withValue(ContactsContract.CommonDataKinds.Phone.NUMBER, phoneNumber)

.withValue(ContactsContract.CommonDataKinds.Phone.TYPE, ContactsContract.CommonDataKinds.Phone.TYPE\_WORK)

.build());

context.getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);

原文链接：https://blog.csdn.net/ucxiii/article/details/59112435