- 基础总结篇之六：ContentProvider之读写联系人

ContentProvider

Android 中，`ContentProvider` 是一种数据封装器，适合在不同进程中共享数据。下面我们来通过两个部分来介绍如何使用 `ContentProvider`

- 使用系统 `ContentProvider`：读取联系人
- 自定义 `ContentProvider`

使用系统 `ContentProvider` 读取联系人

Android 中联系人的数据存储在 `/data/data/com.android.providers.contacts` 下的 databases 下，在开始前我们需要了解 `android.provider.ContactsContract` 这个类，它定义了各种联系人相关的 URL 和每一种类型信息的属性信息。

下面是完整的例子，我们在 androidTest 下新建一个测试用例类 `ContractsReadTest`，完整代码如下：

```
/**
 * @Desc :
 * @Author : Ramon
 * @create 2021/3/12 23:58
 */
@RunWith(AndroidJUnit4::class)
class ContractsReadTest {
    companion object {
        private const val TAG = "ContractsReadTest"

        // content://com.android.contacts/contacts
        private val CONTRACTS_URL: Uri = ContactsContract.Contacts.CONTENT_URI

        // content://com.android.contacts/data/phones
        private val PHONES_URL: Uri = ContactsContract.CommonDataKinds.Phone.CONTENT_URI

        // content://com.android.contacts/data/emails
        private val EMAIL_URI: Uri = ContactsContract.CommonDataKinds.Email.CONTENT_URI

        private val _ID = ContactsContract.Contacts._ID
        private val DISPLAY_NAME = ContactsContract.Contacts.DISPLAY_NAME
        private val HAS_PHONE_NUMBER = ContactsContract.Contacts.HAS_PHONE_NUMBER
        private val CONTACT_ID = ContactsContract.Data.CONTACT_ID

        private val PHONE_NUMBER = ContactsContract.CommonDataKinds.Phone.NUMBER
        private val PHONE_TYPE = ContactsContract.CommonDataKinds.Phone.TYPE
        private val EMAIL_DATA = ContactsContract.CommonDataKinds.Email.DATA
        private val EMAIL_TYPE = ContactsContract.CommonDataKinds.Email.TYPE
    }

    @Test
    fun testReadContracts() {
```

```
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        val contentResolver = appContext.contentResolver
        val c = contentResolver.query(CONTRACTS_URL, null, null, null, null)
        c?.let { cursor ->
            while (c.moveToNext()) {
                val _id = cursor.getInt(cursor.getColumnIndex(_ID))
                val displayName = cursor.getString(cursor.getColumnIndex(DISPLAY_NAME))

                Log.i(TAG, "display name: $displayName")

                // 电话和 email，一个人可能对应多个
                val phones = arrayListOf<String>()
                val emails = arrayListOf<String>()

                // where clause
                val seletion = "$CONTACT_ID=$_id"

                // 获取手机号
                val hasPhoneNumber =
cursor.getInt(cursor.getColumnIndex(HAS_PHONE_NUMBER))
                if (hasPhoneNumber > 0) {
                    val phoneCursor = contentResolver.query(PHONES_URL, null, seletion,
null, null)
                    phoneCursor?.let { pCursor ->
                        while (pCursor.moveToNext()) {
                            val phoneNumber =
pCursor.getString(pCursor.getColumnIndex(PHONE_NUMBER))
                            val phoneType =
pCursor.getInt(pCursor.getColumnIndex(PHONE_TYPE))
                            // 将联系人添加到列表
                            phones.add("${getPhoneTypeNameById(phoneType)}:
$phoneNumber")
                        }
                        pCursor.close()
                    }

                    Log.i(TAG, "phones = $phones")
                }

                // 获取邮箱
                val emCursor = contentResolver.query(EMAIL_URI, null, seletion, null,
null)
                emCursor?.let {emailCursor ->
                    while (emailCursor.moveToNext()) {
                        val emailData =
emailCursor.getString(emailCursor.getColumnIndex(EMAIL_DATA))
                        val emailType =
emailCursor.getInt(emailCursor.getColumnIndex(EMAIL_TYPE))
                        emails.add("${getEmailTypeNameById(emailType)}: $emailData")
                    }
                    emailCursor.close()
                    Log.i(TAG, "emails = $emails")
                }
            }
            cursor.close()
        }
    }
```

```
    private fun getPhoneTypeNameById(typeId: Int): String {
        return when (typeId) {
            ContactsContract.CommonDataKinds.Phone.TYPE_HOME -> "home"
            ContactsContract.CommonDataKinds.Phone.TYPE_MOBILE -> "mobile"
            ContactsContract.CommonDataKinds.Phone.TYPE_WORK -> "work"
            else -> "none"
        }
    }

    private fun getEmailTypeNameById(typeId: Int): String {
        return when (typeId) {
            ContactsContract.CommonDataKinds.Email.TYPE_HOME -> "home"
            ContactsContract.CommonDataKinds.Email.TYPE_WORK -> "work"
            ContactsContract.CommonDataKinds.Email.TYPE_OTHER -> "other"
            else -> "none"
        }
    }
}
```

接下来需要在清单文件中声明读取联系人的权限

```
<!-- 读取联系人 -->
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
```

运行测试用例，在 Log 中可以看到联系人被读出来了。

如果我们是在一个 `Activity` 里读取联系人，可以使用 `ContentResolver` 直接读取，还可以使用 `Activity` 的 `managedQuery` 来读取，来看下这个方法的实现

```
public final Cursor managedQuery(Uri uri,String[] projection,String selection,String[]
selectionArgs,String sortOrder){
        Cursor c = getContentResolver().query(uri, projection, selection, selectionArgs,
sortOrder);
        if (c != null) {
            startManagingCursor(c);
        }
        return c;
}
```

它还是使用了 `ContentResolver` 进行查询操作，但是多了一步 `startManagingCursor` 的操作，它会根据 `Activity` 的生命周期对 `Cursor` 对象进行管理，避免了一些因 `Cursor` 是否释放引起的问题。

向系统 `ContentProvider` 添加联系人

在 AndroidTest 中新建一个测试类 `ContactsWriteTest`，完整代码如下：

```
@RunWith(AndroidJUnit4::class)
class ContactsWriteTest {
    companion object {
```

```
        private const val TAG = "ContactsWriteTest"

        // content://com.android.contacts/raw_contacts
        private val RAW_CONTACTS_URI: Uri = ContactsContract.RawContacts.CONTENT_URI
        // content://com.android.contacts/data
        private val DATA_URI = ContactsContract.Data.CONTENT_URI

        private const val ACCOUNT_TYPE = ContactsContract.RawContacts.ACCOUNT_TYPE
        private const val ACCOUNT_NAME = ContactsContract.RawContacts.ACCOUNT_NAME

        private const val RAW_CONTACT_ID = ContactsContract.Data.RAW_CONTACT_ID
        private const val MIMETYPE = ContactsContract.Data.MIMETYPE

        private const val NAME_ITEM_TYPE =
            ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE
        private const val DISPLAY_NAME =
            ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME

        private const val PHONE_ITEM_TYPE =
            ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE
        private const val PHONE_NUMBER = ContactsContract.CommonDataKinds.Phone.NUMBER
        private const val PHONE_TYPE = ContactsContract.CommonDataKinds.Phone.TYPE
        private const val PHONE_TYPE_HOME =
ContactsContract.CommonDataKinds.Phone.TYPE_HOME
        private const val PHONE_TYPE_MOBILE =
ContactsContract.CommonDataKinds.Phone.TYPE_MOBILE

        private const val EMAIL_ITEM_TYPE =
            ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE
        private const val EMAIL_DATA = ContactsContract.CommonDataKinds.Email.DATA
        private const val EMAIL_TYPE = ContactsContract.CommonDataKinds.Email.TYPE
        private const val EMAIL_TYPE_HOME =
ContactsContract.CommonDataKinds.Email.TYPE_HOME
        private const val EMAIL_TYPE_WORK =
ContactsContract.CommonDataKinds.Email.TYPE_WORK
        private const val AUTHORITY = ContactsContract.AUTHORITY

    }

    @Test
    fun testWriteContact() {
        val operations = arrayListOf<ContentProviderOperation>()

        var operation = ContentProviderOperation.newInsert(RAW_CONTACTS_URI)
            .withValue(ACCOUNT_TYPE, null)
            .withValue(ACCOUNT_NAME, null)
            .build()

        operations.add(operation)

        // 添加联系人名称操作
        operation = ContentProviderOperation.newInsert(DATA_URI)
            .withValueBackReference(RAW_CONTACT_ID, 0)
            .withValue(MIMETYPE, NAME_ITEM_TYPE)
            .withValue(DISPLAY_NAME, "Ramon Lee")
            .build()

        operations.add(operation)
```

```
        // 添加家庭座机号码
        operation = ContentProviderOperation.newInsert(DATA_URI)
            .withValueBackReference(RAW_CONTACT_ID, 0)
            .withValue(MIMETYPE, PHONE_ITEM_TYPE)
            .withValue(PHONE_TYPE, PHONE_TYPE_HOME)
            .withValue(PHONE_NUMBER, "3360075")
            .build()

        operations.add(operation)

        // 添加移动手机号码
        operation = ContentProviderOperation.newInsert(DATA_URI)
            .withValueBackReference(RAW_CONTACT_ID, 0)
            .withValue(MIMETYPE, PHONE_ITEM_TYPE)
            .withValue(PHONE_TYPE, PHONE_TYPE_MOBILE)
            .withValue(PHONE_NUMBER, "15900962200")
            .build()
        operations.add(operation)

        // 添加家庭邮箱
        operation = ContentProviderOperation.newInsert(DATA_URI)
            .withValueBackReference(RAW_CONTACT_ID, 0)
            .withValue(MIMETYPE, EMAIL_ITEM_TYPE)
            .withValue(EMAIL_TYPE, EMAIL_TYPE_HOME)
            .withValue(EMAIL_DATA, "xxxx.gmail.com")
            .build()

        operations.add(operation)

        // 添加工作邮箱
        operation = ContentProviderOperation.newInsert(DATA_URI)
            .withValueBackReference(RAW_CONTACT_ID, 0)
            .withValue(MIMETYPE, EMAIL_ITEM_TYPE)
            .withValue(EMAIL_TYPE, EMAIL_TYPE_WORK)
            .withValue(EMAIL_DATA, "xxxx.ten.com")
            .build()

        operations.add(operation)

        val resolver =
InstrumentationRegistry.getInstrumentation().targetContext.contentResolver
        // 批量执行，返回结果
        val results = resolver.applyBatch(AUTHORITY, operations)
        for (i in results.indices) {
            Log.i(TAG, "result $i = ${results[i]}")
        }
    }
}
```

上面我们把插入联系人的操作分为了几个 `ContentProviderOperation` 来操作，`withValueBackReference(RAW_CONTACT_ID, 0)` 表示引用了第一项操作的 id 值。

遇到一个报错，因为把 Test 方法写到了 Companion object 里面去了...

```
Test class should have exactly one public zero-argument constructor
    at
org.junit.runners.BlockJUnit4ClassRunner.validateZeroArgConstructor(BlockJUnit4ClassRunne
r.java:171)
    at
org.junit.runners.BlockJUnit4ClassRunner.validateConstructor(BlockJUnit4ClassRunner.java:
148)
    at
org.junit.runners.BlockJUnit4ClassRunner.collectInitializationErrors(BlockJUnit4ClassRunn
er.java:127)
    at org.junit.runners.ParentRunner.validate(ParentRunner.java:416)
    at org.junit.runners.ParentRunner.<init>(ParentRunner.java:84)
    at org.junit.runners.BlockJUnit4ClassRunner.<init>(BlockJUnit4ClassRunner.java:65)
```