Content Provider

Learn Android Development For Beginners

Content Providers are one of the primary building blocks that let you share data between applications. Mostly the content provider saves data in the `SQLite` database or in `flat files` at backend. It is the wrapper around the data and Android allows application to expose data source through Content Provider.

Use content provider if you need to share data between the multiple applications. If you don't need to share data among multiple applications then you can use a database directly via `SQLite` Database.

You might have question in your mind that we can directly query database to get the data. But the answer is you can't. A `SQLite` database is private to the application that creates it. It means the one application cannot access the database of other application. Any application can share the data source to other application via Content provider. Content provider is a convenient way to share data with the other application based on structural way.

Android's Built-in Providers

Android has many built in content providers, which are available in the `android.provider`java package. Following are the some built-in content providers.

- Browser
- Call log
- Live Folders
- Contacts Contract
- Media Store
- Settings

Android provides a way to access the built-in content provider in your application. You can perform basic CRUD (Create, Read, Update and Delete) operations on the content providers.

Android also provides a way to create your own content provide by inheriting your class from `ContentProvider` class. This chapter does not cover how to create your own content provider. We will learn about it later.

How to Access Content Provider

Any application can access the data from the Content provider using `ContentResolver` client object. The `ContentResolver` class provide methods to perform basic CRUD (create, Read, Update and Delete) opertations. This mechanism provides an abstraction from the underlying data source and provides standard interface to get data in one application with code running in another application.

Each application `Context` has a single `ContentResolver`, that is accessible uing the `getContentResolver()` method.

```
ContentResolver contentResolver = getContentResolver();
```

The `Context` object should always be available, since the `Activity` and `Service` classes are inherited from `Context` class and `getContentResolver()` method is always available there.

If you are not directly working in `Context` for example in the `Fragment` then first get the `activity` object using `getActivity()` method and call the `getContentResolver()` method to get `ContentResolver` object.

```
ContentResolver contentResolver = getActivity().getContentResolver();
```

The `ContentResolver`provides following methods for CRUD operations.

- **query(Uri, String[], String, String[], String)** returns all the data.
- **insert(Uri, ContentValues)** inserts new data into the content provider
- **update(Uri, ContentValues, String, String[])** updates existing data into the Content Provider.
- **delete(Uri, String, String[])** deletes data from the content provider

`URI` is the important parameter that is being used in all the methods.

Content Provider URIs

It is the most important concept to understand when dealing with Content provider. The methods of `ContentResolver` require `URI`. The URIs for content providers looks like this

```
content://authority/optionalDataPath/optionalId
```

The URI contains four parts.

1. Scheme to Use: For content providers the scheme is always "Content". The "://" is a separator for authority and scheme.

2. Authority: It is the unique for every content provider and it should have same naming convention as the java package. e.g. the Authority name for the Contact Content Provider is "com.android.contacts".

3. Optional Data Path: The optional data path is used to distinguish the data provided by content provider. E.g. ,mediastore content provider distinguish audio, video and images files using different data paths for each of these media types.

4. Optional Id: it is normally used to get the single record. id must be numeric if passed. E.g. if you want to access specific image file.

Almost all `ContentResolver` methods takes the `URI` as parameter, that indicates which content provider it want to access. The `URI` for reading all contacts is given below.

```
content://com.android.contacts/contacts
```

In above URI the “com.android.contacts” is the authority and “contacts” is the optional data path.

Content Provider Access Permission

For accessing the Content Provider the application has to request for specific permissions in manifest file. For example if application requires permission to Get, Add, Update and Delete Contact, add the “`uses-permission`” tag in the manifest file. The contact content provider requires “`READ_CONTACTS`” permission to retrieve data from it and “`WRITE_CONTACTS`” permission for Inserting, Updating and Deleting data.

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

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

Example to Access Contacts using Contact Content Provider

Querying data is most common operation which often used in application. In this example we will user contact provider to get all the contact from your android mobile. Before implementing this example makes sure you have some contact added in your mobile device or emulator (Android Virtual Device). If not then first add some contacts in your device.

Create a new project in Android Studio and name it “**Contact Viewer**”. Open the `AndroidManifest.xml` file and add the “`uses-permission`” tags to get the Read contact permission. The xml of `AndroidManifest.xml` will look like this.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.contactviewer"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="7"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.contactviewer.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

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

</manifest>
```

After getting the permissions in `AndroidManifest.xml`, now set the UI for the application. Add the two `Buttons` and `TextView` to show the Contacts in the layout file. The XML of layout file is below.

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Contacts"
            android:id="@+id/btnContact" />


        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Contacts With Phone"
            android:id="@+id/btnContactWithName" />
    </LinearLayout>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="Click button to load contacts."
        android:id="@+id/txtContacts"

        />


</LinearLayout>
```

In this example Layout file is a Fragment's Layout and we will also write code to fetch the Contacts using contact provider in the Fragment.

In the code file the `btnContact` button fetches the data from `Contact URI` and the second button `btnContactWithName` fetches the data from the `Phone URI`. In the `onCreateView` method of the fragment, set the Layout file and register the click events for the both buttons.

```
  public static class PlaceholderFragment extends Fragment {
        public PlaceholderFragment() {
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                Bundle savedInstanceState) {
            View rootView = inflater.inflate(R.layout.fragment_main, containe


            Button btnContact = (Button)rootView.findViewById(R.id.btnContact)
            Button btnContactWithName = (Button)rootView.findViewById(R.id.btn

            btnContact.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {

                    LoadContacts();
                }
            });

            btnContactWithName.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {

                    LoadContactsWithPhone();
                }
            });

            return rootView;
        }

        .
        .
        .

    }
```

Get Contacts

For getting all the available contacts call the query method of the `ContentResolver` class. The parameters of the query methods are

```
query(URI uri, String[] projection, String selection, String[] selectionArgs,
```

`URI` indicates which content provider you need to access. `Projection` is the names of columns that you want to get. `Selection` is used for filtering the records `SelectionArgs` are values that replaces "?" sign in Selection parameter. `sortOrder` is used for sorting records

`getContacts` method returns the `Cursor` to fetch data one by one. See the code of method below.

```
 private Cursor getContacts() {

     // Get URI
     Uri uri = ContactsContract.Contacts.CONTENT_URI;

     // Set the columns that you want to get from that
     String[] projection = new String[] {
                                    ContactsContract.Contacts._ID,
                                    ContactsContract.Contacts.DISPLAY_NAM

     // Set Where clause
     String selection = ContactsContract.Contacts.IN_VISIBLE_GROUP + " = '" +

     String[] selectionArgs = null;

     String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " COLLATE LOC

     ContentResolver cr = getActivity().getContentResolver() ;
     return cr.query(uri, projection, selection, selectionArgs, sortOrder);

 }
```

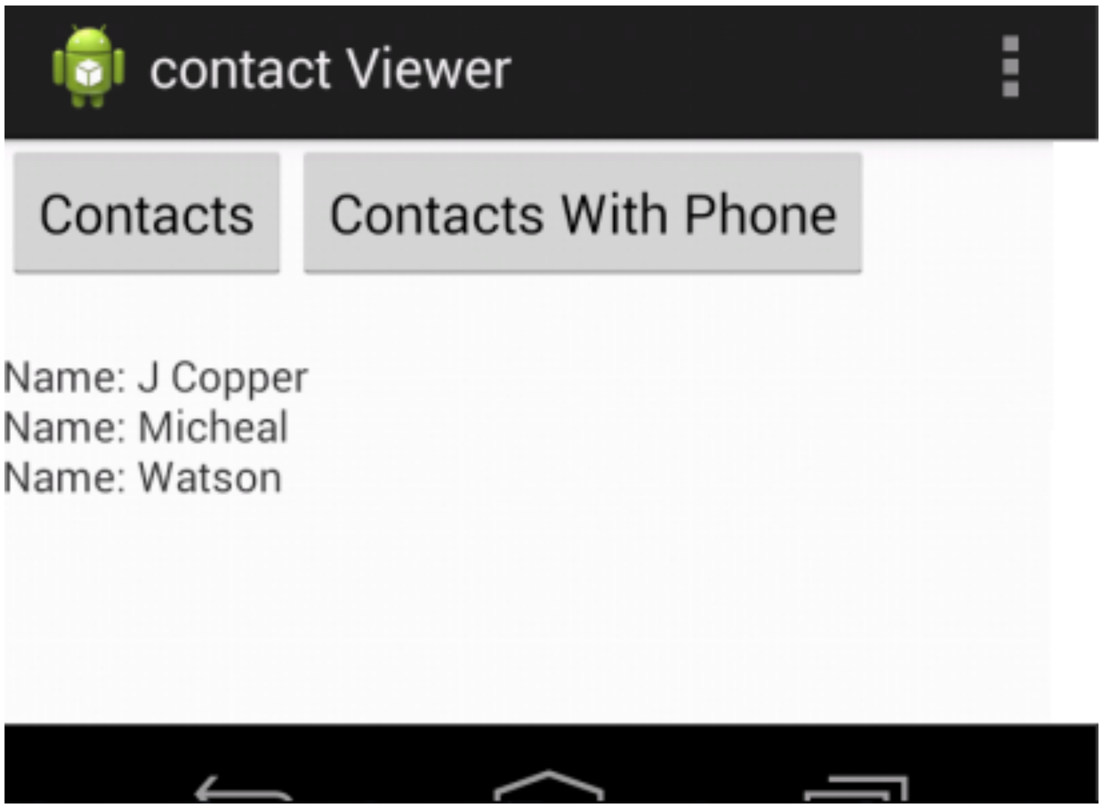
Get the contact providers `URI` using the `ContactsContract.Contacts.CONTENT_URI` and the set the name of columns that you want to get in `projection` variable. The `ContactsContract.Contacts._ID`, ContactsContract.Contacts.DISPLAY_NAME are the constants that contains the column name. Similarly also assign values to other variables those are required for query method.

Get the ContentResolver using getActivity().getContentResolver() statement and call the query method that returns the Cursor to read record one by one.

The LoadContacts method gets the Cursor using getContact method, read all records one by one and appends it into the TextView control that is available on the layout.

```
private void LoadContacts()
{
   TextView contactView = (TextView) getView().findViewById(R.id.txtContacts)
   contactView.setText("");

   Cursor cursor = getContacts();

   while (cursor.moveToNext()) {

       String displayName = cursor.getString(cursor
                                  .getColumnIndex(ContactsContract.Data.DISPLA

       contactView.append("Name: ");
       contactView.append(displayName);
       contactView.append("\n");

    }

}
```

The cursor.moveToNext() return true if it has next record otherwise it returns false. For getting the data for the any column first get the index of the column using cursor.getColumnIndex(ContactsContract.Data.DISPLAY_NAME) and then get the data according to type using `getString()`, `getFloat()` etc methods. Now the output of the program will look like this.



```
Name: J Copper
Name: Micheal
Name: Watson
```

Get Contacts with Phone No

Contact Content Provider provides some other URIs to get some extra data. The `ContactsContract.CommonDataKinds.Phone.CONTENT_URI` returns the data with phone number. Add the `ContactsContract.CommonDataKinds.Phone.NUMBER` column in the `projection` and set the `selection` and `sortOrder` parameters accordingly.

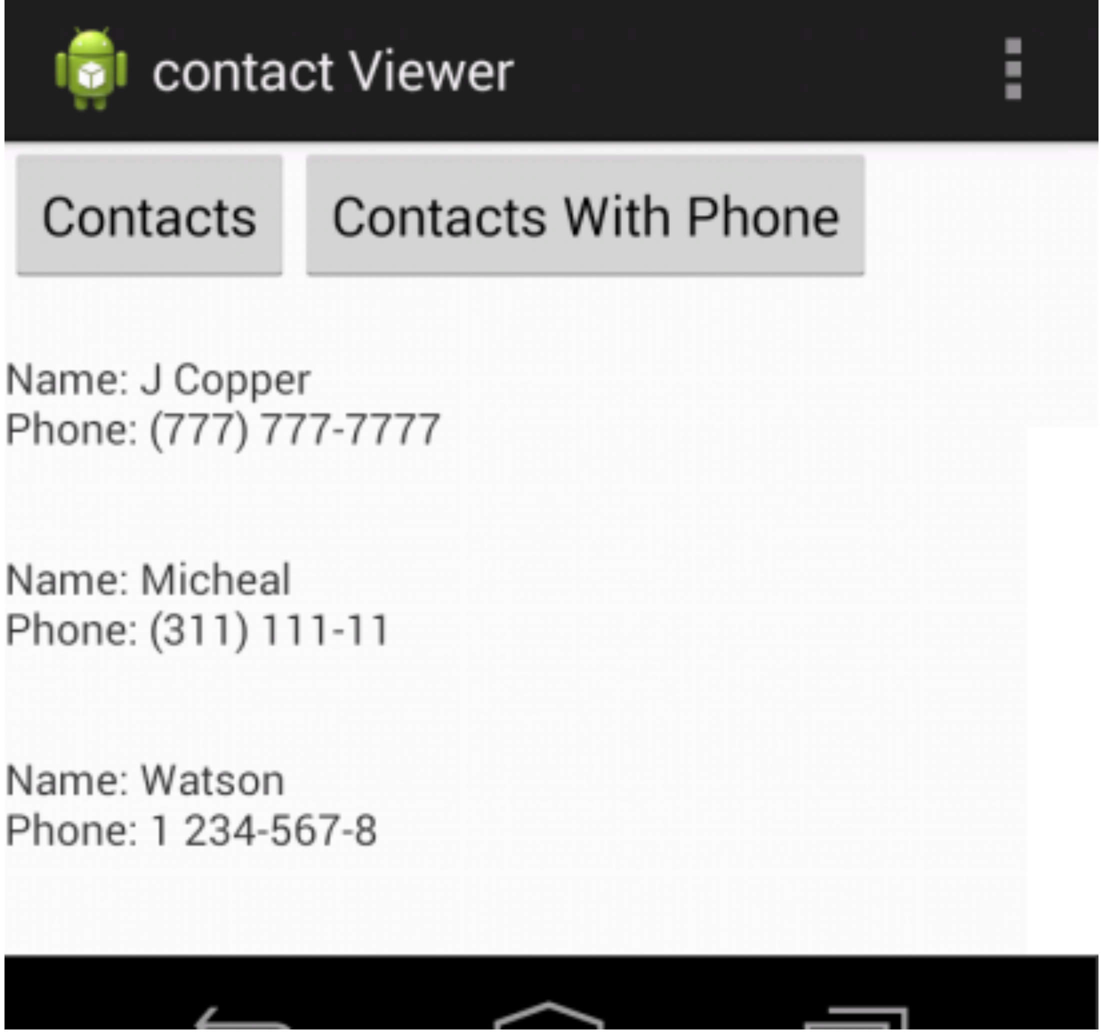
```
private Cursor getContactsWithPhone()
{
  Uri uri = ContactsContract.CommonDataKinds.Phone.CONTENT_URI;
  String[] projection = new String[] { ContactsContract.Contacts._ID,
                 ContactsContract.Contacts.DISPLAY_NAME,
                 ContactsContract.CommonDataKinds.Phone.NUMBER
                 };

  String selection = ContactsContract.Contacts.HAS_PHONE_NUMBER  + " = ?";

  String[] selectionArgs = new String[]{ String.valueOf(1)};

  String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " COLLATE LOCAI
```

```
   return getActivity().getContentResolver().query(uri, projection, selection,
 }
```

After getting the `Cursor` iterate the record one by one and get the information for each column using `getString()` method of the cursor.

```
 private void LoadContactsWithPhone()
 {
    TextView contactView = (TextView) getView().findViewById(R.id.txtContacts)
    contactView.setText("");

    Cursor cursor = getContactsWithPhone();

    while (cursor.moveToNext()) {

        String displayName = cursor.getString(cursor
                        .getColumnIndex(ContactsContract.Data.DISPLAY_NAME));

        String phoneNo = cursor.getString(cursor
                        .getColumnIndex(ContactsContract.CommonDataKinds.Phone

        contactView.append("Name: ");
        contactView.append(displayName);
        contactView.append("\n");

        contactView.append("Phone: ");
        contactView.append(phoneNo);
        contactView.append("\n\n");

    }

 }
```

The out of the above code will look like this.



Name: J Copper
Phone: (777) 777-7777

Name: Micheal
Phone: (311) 111-11

Name: Watson
Phone: 1 234-567-8

In the next chapters later we will learn about how to create our own content provider and how can we perform Insert, update and delete operations later.

Download Source Code

Click Here to Download Source Code