Hướng dẫn lập trình cơ bản với Android - Bài 7 : Content Provider
Reflink: http://diendan.vietandroid.com/cac-bai-hoc-co-kem-ma-nguon/1107-huong-dan-lap-trinh-co-ban-voi-android-bai-7-content-provider.html

List tutorial

Bài 7 : Android Content Provider

Cuối tuần rảnh rỗi hơn, nhớ ra là phải tiếp tục viết Tutorial cho diễn đàn, rất nhiều các Newbies đang cần

Trước khi vào bài, các bạn down Sourcecode của bài học tại đây
Sourcecode của bài học bao gồm 3 Project demo :
ContentProviderDemo
ContentProviderDemo1
ContentProviderDemo2

Mỗi Demo sẽ giải quyết từng vấn đề của Tutorial. Các bạn cứ add hết vào workspace để tiện theo dõi.

1. Giới thiệu Content Provider

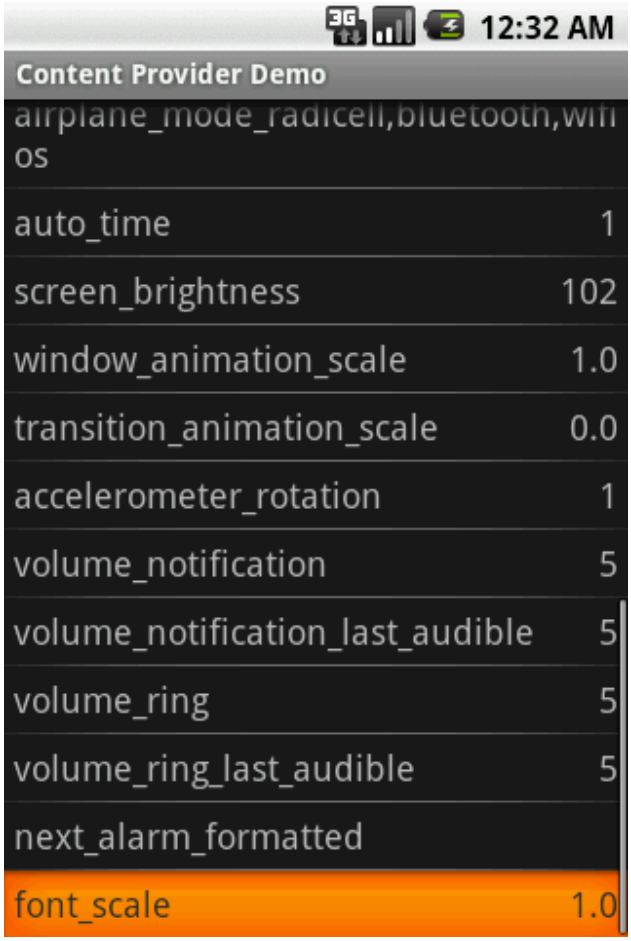
Content Provider là 1 trong 4 thành phần cơ bản của 1 ứng dụng Android thường có bao gồm:
1. Activity
2. Service
3. Broadcast Receiver
4. Content Provider
Một Content Provider cung cấp một tập chi tiết dữ liệu ứng dụng đến các ứng dụng khác.
Thường được sử dụng khi chúng ta muốn tạo cơ sở dữ liệu dưới dạng public (các ứng dụng khác có thể truy xuất).
Dữ liệu thường được lưu trữ ở file hệ thống, hoặc trong một SQLite database.
Đơn giản để các bạn có thể hình dung như : Danh bạ, Call log, cấu hình cài đặt...trên điện thoại là dữ liệu dưới dạng Content Provider.

Content Provider hiện thực một tập phương thức chuẩn mà các ứng dụng khác có thể truy xuất và lưu trữ dữ liệu của loại nó điều khiển.
Tuy nhiên, những ứng dụng không thể gọi các phương thức trực tiếp. Hơn thế chúng dùng lớp Content Resolver và gọi những phương thức đó. Một Content Resolver có thể giao tiếp đến nhiều content provider; nó cộng tác với các provider để quản lý bất kỳ giao tiếp bên trong liên quan.

Đơn giản hơn, chúng ta có thể làm 1 ứng dụng nhỏ để lấy tất cả các thông tin cấu hình trong

máy load lên listview. Các bạn có thể chạy Project ContentProviderDemo1 trong SourcecodeDemo.


12:31 AM
Content Provider Demo
volume_music 11
volume_system 5
volume_voice 4
volume_alarm 6
mode_ringer 2
vibrate_on 4
mode_ringer_streams_affected 38
mute_streams_affected 46
dim_screen 1
stay_on_while_plugged_in 1
screen_off_timeout 60000
emergency_tone 0



Chúng ta có thể tìm hiểu sơ qua về code của demo này, rất ngắn gọn

Mã:

```
      ContentResolver cr = getContentResolver();
      Cursor cursor = cr.query(Settings.System.CONTENT_URI, null, null,
null, null);
      startManagingCursor(cursor);

      ListView listView = (ListView) findViewById(R.id.listView);
      String[] from = { Settings.System.NAME, Settings.System.VALUE };
      int[] to = { R.id.textName, R.id.textValue };
      SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
R.layout.row, cursor, from, to);
      listView.setAdapter(adapter);
```

Như các bạn thấy, chỉ cần 2 dòng code đơn giản để lấy được con trỏ thao tác trên tập dữ liệu cần lấy:

Mã:

```
      ContentResolver cr = getContentResolver();
      Cursor cursor = cr.query(Settings.System.CONTENT_URI, null, null,
null, null);
```

Lớp Content Resolver cung cấp các phương thức xử lý dữ liệu thông qua các Uri, mỗi Content Provider có 1 Uri cụ thể , ở đây Uri *Settings.System.CONTENT_URI*sẽ trả lại tập dữ liệu là thông tin cấu hình của thiết bị.
Sau khi lấy được con trỏ tới tập dữ liệu, việc còn lại đơn giản là bind data lên listview để hiển thị:
Mã:

```
       startManagingCursor(cursor);
      ListView listView = (ListView) findViewById(R.id.listView);
      String[] from = { Settings.System.NAME, Settings.System.VALUE };
      int[] to = { R.id.textName, R.id.textValue };
      SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
R.layout.row, cursor, from, to);
      listView.setAdapter(adapter)
```

2. Tạo và sử dụng 1 Content Provider do người dùng tự định nghĩa

Để dễ hiểu hơn các bạn mở Project ContentProviderDemo trong Sourcecode đã down về. Trong Project đó mình tạo 1 Content Provider Books, mỗi bản ghi Book bao gồm 2 trường : ID và Title.

Sau đây là các bước để tạo 1 Content Provider cơ bản (cụ thể là tạo ContentProvider Book)

1. Tạo 1 class thừa kế lớp ContentProvider
Mã:

```
public class BookProvider extends ContentProvider
```

2. Định nghĩa 1 biến Uri (public static final) được gọi CONTENT_URI. Các xâu này luôn được bắt đầu bằng "content://" tiếp theo đó là nội dung của mà ContentProvider xử lý. Xâu này phải có đặc tính là duy nhất.
Mã:

```
       public static final String PROVIDER_NAME =
"com.vietandroid.provider.Books";
       public static final Uri CONTENT_URI = Uri.parse("content://" +
PROVIDER_NAME + "/books");
```

3. Khai báo các xâu để định nghĩa cho từng thuộc tính tương ứng với các cột giá trị từ Cursor.
Mã:

```
       public static final String _ID = "_id";
       public static final String TITLE = "title";
```

4. Chúng ta cần tạo hệ thống chứa dữ liệu cho ContentProvider, có thể chưa dưới nhiều hình thức : sử dụng XML, thông qua CSDL SQLite, hay thậm chí là WebService. Trong Demo này chúng ta sử dụng cách phổ biến nhất đó là SQLite:

Mã:

```
       private SQLiteDatabase bookDB;
       private static final String DATABASE_NAME = "Books";
       private static final String DATABASE_TABLE = "titles";
       private static final int DATABASE_VERSION = 1;
```

5. Định nghĩa tên của các cột mà chúng ta sẽ trả lại giá trị cho các clients.Nếu chúng ta đang sử dụng Database ContentProvider hay các lớp SQLiteOpenHelper, tên các cột này chính là id của các cột trong cơ sở dữ liệu SQL. Trong trường hợp này, chúng ta phải gộp cả cột có giá trị là số nguyên được gọi "_id" để định nghĩa id của mỗi bản ghi. Nếu đang sử dụng cơ sở dữ liệu SQLite, nó sẽ là INTEGER PRIMARY KEY AUTOINCREMENT. Tùy chọn AUTOINCREMENT không bắt buộc, có tác dụng tự động tăng ID của mỗi bản ghi lên nếu người dùng không nhập. Android cung cấp

SQLiteOpenHelper giúp tạo và quản lý các phiên bản của cơ sở dữ liệu.

Mã:

```
private static final String DATABASE_CREATE =
                "create table " + DATABASE_TABLE +
                " (_id integer primary key autoincrement, "
                + "title text not null);";

private static class DatabaseHelper extends SQLiteOpenHelper
        {
                public DatabaseHelper(Context context) {
                        super(context, DATABASE_NAME , null,
DATABASE_VERSION);
                }

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

                @Override
                public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion) {
                        db.execSQL("DROP TABLE IF EXISTS titles");
                        onCreate(db);
                }
        }
```

6. Nếu chúng ta muốn public các dữ liệu kiểu byte như bitmap thì các trường mà chứa dữ liệu này nên là một xâu với 1 content://URI cho file đó. Đây chính là liên kết để các ứng dụng khác có thể truy cập và sử dụng dữ liệu bitmap này.

7. Sử dụng Cursor để thao tác trên tập dữ liệu : query (), update(), insert(), delete()….. Có thể gọi phương thức ContentResolver.notifyChange() để biếtkhi nào dữ liệu được cập nhật.

Add Book

Mã:

```
        @Override
        public Uri insert(Uri uri, ContentValues values) {
                long rowID = bookDB.insert(DATABASE_TABLE, "", values);
                if(rowID > 0)
                {
                        Uri mUri =
ContentUris.withAppendedId(CONTENT_URI, rowID);

        getContext().getContentResolver().notifyChange(mUri, null);
                        return mUri;

                }
                throw new SQLException("Failed to insert new row into "
+ uri);
        }
```

Get All Books

Mã:

```
        @Override
        public Cursor query(Uri uri, String[] projection, String
selection,
                        String[] selectionArgs, String sortOrder) {
                SQLiteQueryBuilder sqlBuilder = new
SQLiteQueryBuilder();
                sqlBuilder.setTables(DATABASE_TABLE);
                if(uriMatcher.match(uri) == BOOK_ID)
                        sqlBuilder.appendWhere(_ID + "=" +
uri.getPathSegments().get(1));
                if(sortOrder == null || sortOrder == "")
                        sortOrder = TITLE;
                Cursor c = sqlBuilder.query(bookDB, projection,
selection, selectionArgs, null, null, sortOrder);
                c.setNotificationUri(getContext().getContentResolver(),
uri);
                return c;
        }
}
```

Mình chỉ demo 2 chức năng là thêm sách và lấy toàn bộ bản ghi trong CSDL , ngoài ra các phương thức edit, sửa , update, xóa... các bạn có thể tự làm .

8. Khai báo Content Provider trong file AndroidManifest.xml
Mã:

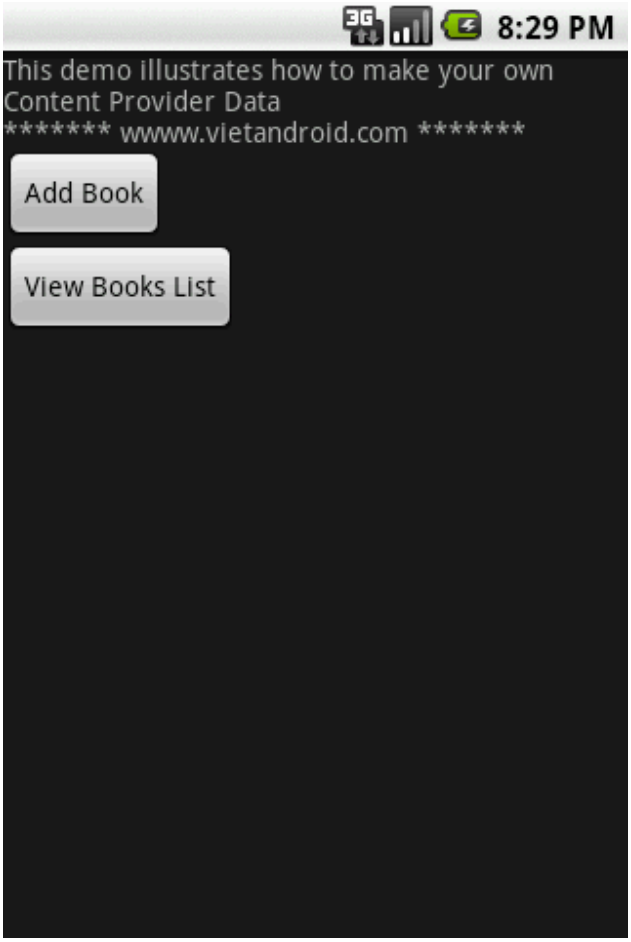
```
                <provider android:name = "BookProvider"

        android:authorities="com.vietandroid.provider.Books" />
```

Như vậy chúng ta đã tạo xong ContentProvider Book tự định nghĩa.

9. Test thử thành quả :
Mỗi content Provider gắn với 1 Uri cụ thể, như trên thì ContentProvider Book có Uri là:
Mã:

```
com.vietandroid.provider.Books/books
```

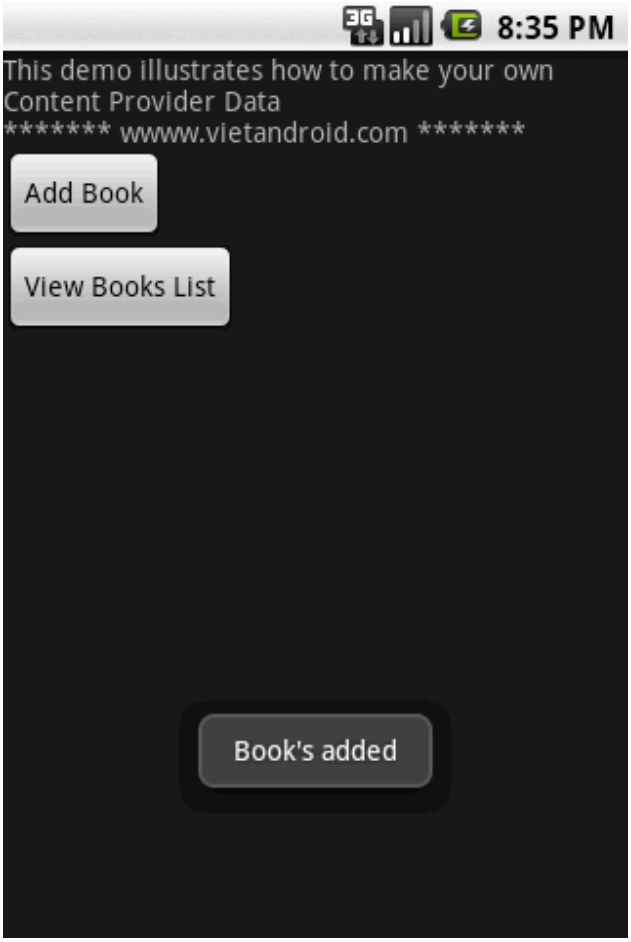
Để test thử , vẫn trong Project Demo ContentProviderDemo , các bạn có thể thêm 2 Button Add Book và View All Books vào . Giao diện như sau:



Chúng ta sẽ thêm 1 Book vào CSDL thông qua URI này:
Mã:

```
       public void addBook(String title)
    {
        ContentValues values = new ContentValues();
        values.put(BookProvider.TITLE, title);
        Uri uriInsert =
getContentResolver().insert(BookProvider.CONTENT_URI, values);
        if(uriInsert != null)
        {
                Toast.makeText(this, "Book's added",
Toast.LENGTH_SHORT).show();
        }
        Log.d(getClass().getSimpleName(),uriInsert.toString());
    }
```

Kết quả :



Truy vấn toàn bộ dữ liệu Books có trong CSDL
Mã:

```
    public void getAllBooks()
    {
        Uri uriGetListTitles =
Uri.parse("content://com.vietandroid.provider.Books/books");
        Cursor c = managedQuery(uriGetListTitles, null, null, null,
"title desc");
        if(c.moveToFirst()){
                do{
                        String bookRecord = "ID = " +
c.getString(c.getColumnIndex(BookProvider._ID)) + " Title = " +

        c.getString(c.getColumnIndex(BookProvider.TITLE));
                        Toast.makeText(this, bookRecord ,
Toast.LENGTH_LONG).show();
                }while(c.moveToNext());
        }
    }
```

Kết quả :
[IMGhttp://i123.photobucket.com/albums/o286/firewall7845/VietAndroid/2-1.png[/IMG]

3. Sử dụng dữ liệu Content Provider từ 1 ứng dụng bất kỳ

Ở Bài 6 mình đã đề cập về cơ sở dữ liệu SQLite Database, dạng dữ liệu này không public cho các ứng dụng khác sử dụng, dữ liệu của ứng dụng nào thì ứng dụng đó sử dụng.
1 lợi thế của dữ liệu dưới dạng Content Provider là public, tất cả các ứng dụng đều có thể truy cập và sử dụng.

Phần này các bạn sử dụng ProjectDemo là ContentProviderDemo2 trong sourcecode đi kèm ban đầu.
Demo này chỉ đơn giản là đọc lại toàn bộ dữ liệu trong CSDL Books được tạo trong phần 2.
Như mình đã nói ở trên, chỉ cần lấy được Uri của ContentProvider cần lấy và các tên của các trường dữ liệu thì chúng ta có thể truy vấn được hết.
Trong hàm onCreate() các bạn thêm vào:
Mã:

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
       Uri uriGetListTitles =
Uri.parse("content://com.vietandroid.provider.Books/books");
        Cursor c = managedQuery(uriGetListTitles, null, null, null,
"title desc");
        if(c != null)
        {
                if(c.moveToFirst()){
                        do{
                                String bookRecord = "ID = " +
c.getString(c.getColumnIndex("_id")) + " Title = " +

        c.getString(c.getColumnIndex("title"));
                                Toast.makeText(this, bookRecord ,
Toast.LENGTH_LONG).show();
                        }while(c.moveToNext());
                }
        }
        else {
                Toast.makeText(this, "Database is emtpy",
Toast.LENGTH_SHORT).show();
        }
    }
```

2 trường dữ liệu ở đây được định nghĩa ở trên là "_id" và "title". Phần truy vấn vẫn như vậy.
Kết quả :

==========H
ẾT==========

Như vậy, mình đã trình bày Tổng quan và rất chi tiết về Content Provider , cách sử dụng cũng không khó. Mọi thắc mắc trong quá trình học bài này các bạn có thể post ở dưới , mình sẽ trả lời sớm nhất có thể

Bài tiếp theo sẽ là Thread & Handler.