# ContentProvider 实验

## 一、 实验目的

- 掌握 android 的四大组件之一 ContentProvider 使用；
- 掌握 Uri 使用；
- 复习 SqliteDatabase 的内容

## 二、实验条件

- PC 机
- JDK8
- Android Studio

## 三、实验原理

ContentProvider 是 Android 应用程序的四大基本组件之一。它主要用于管理 APP 共享的数据，通过提供一套标准化的统一接口来简化异构数据的共享访问问题。ContentProvider 将 APP 可访问的存在文件系统、SQLite 数据库、Web 或持久性存储方案中的数据等进行转换，并通过统一的标准接口提供给其他应用。Android 内置的许多数据都通过 ContentProvider 提供，例如视频，音频，图片，通讯录等。其他的应用通过 ContentProvider 可以根据权限对这些共享数据进行增删改查操作。

ContentProvider 的使用场景

- 希望提供复杂的数据或者文件给其他程序
- 希望让用户从应用中复制复杂数据给其他程序
- 希望使用搜索框架时提供自定义的搜索提示

ContentProvider 能提供两种数据类型的数据

- 文件数据

诸如图片，音频，或者视频等信息的文件数据。文件存储在应用的私有空间。其他应用请求共享的文件数据时可用 ContentProvider 提供文件的句柄.

- "格式化"数据

这类数据通常存入数据库、数组、或者类似结构。储存这种数据的常见方式是用 SQLite 数据库，当然其他任何类型的持久性存储方案也可以使用。

当应用通过 ContentProvider 对外共享数据时，需要处理两件事，一是实现一个 ContentProvider 类，其二是定义 URI。

ContentProvider 类的实现需要继承 ContentProvider 抽象类并重写下面方法：

- onCreate()
- query(Uri, String[], String, String[], String)
- insert(Uri, ContentValues)
- update(Uri, ContentValues, String, String[])
- delete(Uri, String, String[])

- getType(Uri)

定义 URI

**Uri**

统一资源标识符（Uniform Resource Identifier，URI)是一个用于标识某个互联网资源名称的字符串。 该种标识允许用户对网络中的资源通过特定的协议进行交互操作。URI 由包括确定语法和相关协议的方案所定义。URI 文法由 URI 协议名（例如“http”,“ftp”,“mailto”或“file”），一个冒号，和协议对应的内容所构成。特定的协议定义了协议内容的语法和语义。ContentProvider 中的 URI 由四部分组成：

- Scheme：在 ContentProvider 中，它由“content:”标识。
- Authority：ContentProvider 用 Authority 作为内部名称。为了避免冲突，通常应该用完整的包名来定义扩展。例如 Android 包名是 com.example.<appname>, Authority 可以设置为 com.example.<appname>.provider.
- Location：通常在 Authority 后附加指向单独表的路径的方式创建 Location。 例如，两个表 table1 和 table2，结合前面的例子的 Authority 可生成 Content URIs com.example.<appname>.provider/table1 和 com.example.<appname>.provider/table2。
- Query：这部分用于定位特定记录的数据

**UriMatcher**

因为 Uri 代表了要操作的数据，所以我们经常需要解析 Uri，并从 Uri 中获取数据。Android 系统提供了两个用于操作 Uri 的工具类，分别为 UriMatcher 和 ContentUris 。掌握它们的使用，会便于我们的开发工作。

UriMatcher 类用于匹配 Uri，它的用法如下：

首先第一步把你需要匹配 Uri 路径全部给注册上，如下：

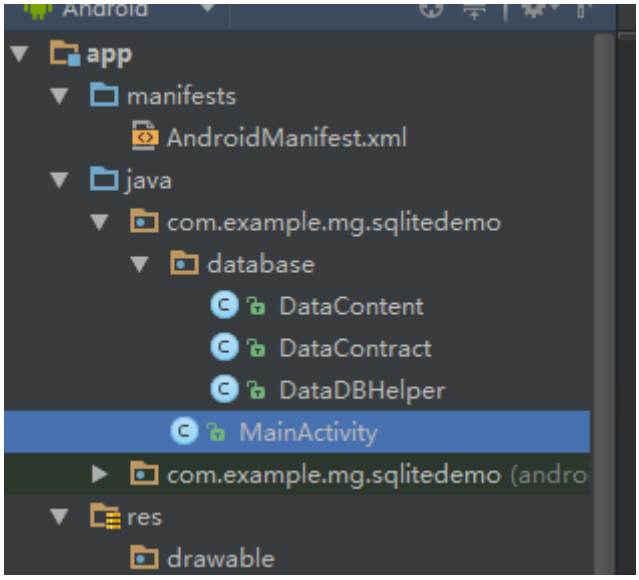
```
// 常量 UriMatcher.NO_MATCH 表示不匹配任何路径的返回码
UriMatcher sMatcher = new UriMatcher(UriMatcher.NO_MATCH);
//如果 match()方法匹配 content://com.ljq.provider.personprovider/person 路径，返回匹配码为 1
sMatcher.addURI("com.ljq.provider.personprovider", "person", 1);//添加需要匹配 uri，如果匹配就会返回匹配码
//如果 match()方法匹配 content://com.ljq.provider.personprovider/person/230 路径，返回匹配码为 2
sMatcher.addURI("com.ljq.provider.personprovider", "person/#", 2);//#号为通配符
switch (sMatcher.match(Uri.parse("content://com.ljq.provider.personprovider/person/10"))) {
 case 1
  break;
    case 2
  break;
  default://不匹配
   break
```

```
    }
```

注册完需要匹配的 Uri 后，就可以使用 sMatcher.match(uri)方法对输入的 Uri 进行匹配，如果匹配就返回匹配码，匹配码是调用 addURI()方法传入的第三个参数，假设匹配 content://com.ljq.provider.personprovider/person 路径，返回的匹配码为 1

## 四、实验步骤

本实验是在实验二 sqlitedatabase 的基础上实现的(蓝色部分为新增实验内容)。工程文件结构如下:




**AndroidMainfest.xml**

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

    <application
        ......
        <activity
            .....
        </activity>
        //authorities 是包名，name 是创建的 contentProvider 的继承类
        <provider
            android:authorities="com.example.mg.sqlitedemo"
            android:name=".database.DataContent"/>
    </application>
</manifest>
```

**DataContract.java**

```
......
```

```
//注意，这边的变量值不能带有空格！比如 public static final String COULUMN_PHONE_NUMBER = "phone number";就是错误的。
public class DataContract {


    public static final String CONTENT_AUTHORITY = "com.example.mg.sqlitedemo";
    public static final String PATH_CONTACTS = "datademo";

    public static final class DataEntry implements BaseColumns{

        //创建一个Uri
        //可以使用这个
       //      public      static      final      Uri      CONTENT_URI      =
Uri.parse("content://com.example.mg.sqlitdemo/datademo");
                //Uri.parse("content://" + CONTENT_AUTHORITY + "/datademo");
        //或者下面代码
        // CONTENT_TYPE 用于gettype()返回的是一列 还是多列

        public  static  final  Uri  CONTENT_URI  =  Uri.parse("content://"  +
CONTENT_AUTHORITY + "/" + PATH_CONTACTS);
        public static final String CONTENT_TYPE = "vnd.android.cursor.dir/" +
CONTENT_AUTHORITY + "/" + PATH_CONTACTS;
        public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/"
+ CONTENT_AUTHORITY + "/" + PATH_CONTACTS;


        public static final String TABLE_NAME = "datademo";
        public static final String COULUMN_USER_NAME = "name";
        public static final String COULUMN_PHONE_NUMBER = "phonenumber";
    }
}
```

**DataContent.java**

```
public class DataContent extends ContentProvider{

    private DataDBHelper dbOpenHelper;

    private final static UriMatcher sUriMatcher = buildUriMatcher();
    private static final int DATA = 100;

    private static UriMatcher buildUriMatcher(){
        final UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
        final String authority = DataContract.CONTENT_AUTHORITY;
```

```
        matcher.addURI(authority,DataContract.PATH_CONTACTS,DATA);

        return matcher;
    }



    @Override
    public boolean onCreate() {
        dbOpenHelper = new DataDBHelper(getContext());
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] strings, String s, String[] strings2,
String s2) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues contentValues) {

        SQLiteDatabase db = dbOpenHelper.getWritableDatabase();
        long id = 0;
        Uri returnUri;
        switch (sUriMatcher.match(uri)){
            case DATA: {
                id    =    db.insert(DataContract.DataEntry.TABLE_NAME,    null,
contentValues);
                if (id > 0)
                   returnUri = ContentUris.withAppendedId(uri, id);
                else
                    throw new android.database.SQLException("Failed to insert row
into " + uri);
                break;
            }
            default:
                throw new UnsupportedOperationException("Unknown uri: " + uri);
        }
        return returnUri;
    }

    @Override
```

```
    public int update(Uri uri, ContentValues contentValues, String s, String[]
strings) {
        return 0;
    }

    @Override
    public int delete(Uri uri, String s, String[] strings) {
        return 0;
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }
}
```

可以通过 re 文件查看器看到创建的数据库（根目录下的 data/data/com.example.mg.sqlitedemo/databases/DataDemo/datademo）

## 五、实验报告要求

实验报告中要包含以下几个部分：

1、实验目的
2、实验条件
3、实验原理
4、实验步骤分析
5、实验结果与总结
6、实验思考题

实验步骤要详细，关键步骤要有截图，运行结果也要有截图。

## 六、实验思考题

1、修改程序，增加 query()，delete()的用法

**参考资料**


http://developer.android.com/guide/topics/providers/content-providers.html
http://developer.android.com/guide/topics/providers/content-provider-basics.html
http://developer.android.com/guide/topics/providers/content-provider-creating.html
http://developer.android.com/guide/components/fundamentals.html
http://developer.android.com/intl/zh-cn/reference/android/content/ContentProvider.html