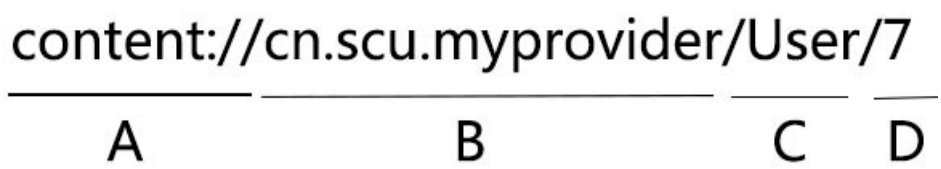
1.ContentProvider是什么？

ContentProvider（内容提供者）是Android的四大组件之一，管理android以结构化方式存放的数据，以相对安全的方式封装数据（表）并且提供简易的处理机制和统一的访问接口供**其他程序**调用。 Android的数据存储方式总共有五种，分别是：Shared Preferences、网络存储、文件存储、外储存储、SQLite。但一般这些存储都只是在单独的一个应用程序之中达到一个数据的共享，有时候我们需要操作其他应用程序的一些数据，就会用到ContentProvider。而且Android为常见的一些数据提供了默认的ContentProvider（包括音频、视频、图片和通讯录等）。

但注意ContentProvider它也只是一个中间人，真正操作的数据源可能是数据库，也可以是文件、xml或网络等其他存储方式。

2 .URL

URL（统一资源标识符）代表要操作的数据，可以用来标识每个ContentProvider，这样你就可以通过指定的URI找到想要的ContentProvider,从中获取或修改数据。 在Android中URI的格式如下图所示：



- A schema，已经由Android所规定为：content://.
- B

主机名（Authority），是URI的授权部分，是唯一标识符，用来定位ContentProvider。

> C部分和D部分：是每个ContentProvider内部的路径部分

- C

指向一个对象集合，一般用表的名字，如果没有指定D部分，则返回全部记录。

- D

指向特定的记录，这里表示操作user表id为7的记录。如果要操作user表中id为7的记录的name字段， D部分变为 **/7/name**即可。

> URI模式匹配通配符
>
> *：匹配的任意长度的任何有效字符的字符串。
>
> #：匹配的任意长度的数字字符的字符串。
>
> 如：
>
> content://com.example.app.provider/* 匹配provider的任何内容url
>
> content://com.example.app.provider/table3/# 匹配table3的所有行

2.1 MIME

MIME是指定某个扩展名的文件用一种应用程序来打开，就像你用浏览器查看PDF格式的文件，浏览器会选择合适的应用来打开一样。Android中的工作方式跟HTTP类似，ContentProvider会根据URI来返回MIME类型，ContentProvider会返回一个包含两部分的字符串。MIME类型一般包含两部分，如：

> text/html text/css text/xml application/pdf

分为类型和子类型，Android遵循类似的约定来定义MIME类型，每个内容类型的Android MIME类型有两种形式：多条记录（集合）和单条记录。

集合记录：

```
vnd.android.cursor.dir/自定义
```

单条记录：

```
vnd.android.cursor.item/自定义
```

vnd表示这些类型和子类型具有非标准的、供应商特定的形式。Android中类型已经固定好了，不能更改，只能区别是集合还是单条具体记录，子类型可以按照格式自己填写。　　在使用Intent时，会用到MIME，根据Mimetype打开符合条件的活动。

下面分别介绍Android系统提供了两个用于操作Uri的工具类：ContentUris和UriMatcher。

2.2 ContentUris

ContetnUris包含一个便利的函数withAppendedId()来向URI追加一个id。

```
Uri uri = Uri.parse("content://cn.scu.myprovider/user")
Uri resultUri = ContentUris.withAppendedId(uri, 7);

//生成后的Uri为: content://cn.scu.myprovider/user/7
```

同时提供parseId(uri)方法用于从URL中获取ID:

```
Uri uri = Uri.parse("content://cn.scu.myprovider/user/7")
long personid = ContentUris.parseId(uri);
//获取的结果为:7
```

2.3 UriMatcher

UriMatcher本质上是一个文本过滤器，用在contentProvider中帮助我们过滤，分辨出查询者想要查询哪个数据表。

举例说明：

- 第一步，初始化：

```
UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
//常量UriMatcher.NO_MATCH表示不匹配任何路径的返回码
```

- 第二步，注册需要的Uri：

```
//USER 和 USER_ID是两个int型数据
matcher.addURI("cn.scu.myprovider", "user", USER);
matcher.addURI("cn.scu.myprovider", "user/#",USER_ID);
//如果match()方法匹配content://cn.scu.myprovider/user路径，返回匹配码为USER
```

- 第三部，与已经注册的Uri进行匹配:

```
/*
     * 如果操作集合，则必须以vnd.android.cursor.dir开头
     * 如果操作非集合，则必须以vnd.android.cursor.item开头
     * */
    @Override
    public String getType(Uri uri) {
    Uri uri = Uri.parse("content://" + "cn.scu.myprovider" + "/user");
        switch(matcher.match(uri)){
        case USER:
            return "vnd.android.cursor.dir/user";
        case USER_ID:
            return "vnd.android.cursor.item/user";
        }
    }
```

3.ContentProvider的主要方法

> public boolean onCreate()

ContentProvider创建后　或　打开系统后其它应用第一次访问该ContentProvider时调用。

> public Uri insert(Uri uri, ContentValues values)

外部应用向ContentProvider中添加数据。

> public int delete(Uri uri, String selection, String[] selectionArgs)

外部应用从ContentProvider删除数据。

> public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)：

外部应用更新ContentProvider中的数据。

> public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)

供外部应用从ContentProvider中获取数据。

> public String getType(Uri uri)

该方法用于返回当前Url所代表数据的MIME类型。

4 .ContentResolver

ContentResolver通过URI来查询ContentProvider中提供的数据。除了URI以 外，还必须知道需要获取的数据段的名称，以及此数据段的数据类型。如果你需要获取一个特定的记录，你就必须知道当前记录的ID，也就是URI中D部分。

ContentResolver 类提供了与ContentProvider类相同签名的四个方法：

> public Uri insert(Uri uri, ContentValues values)　//添加
>
> public int delete(Uri uri, String selection, String[] selectionArgs)　//删除
>
> public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)　//更新
>
> public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)//获取

实例代码：

```
ContentResolver resolver =  getContentResolver();
Uri uri = Uri.parse("content://cn.scu.myprovider/user");

//添加一条记录
ContentValues values = new ContentValues();
values.put("name", "fanrunqi");
values.put("age", 24);
resolver.insert(uri, values);

//获取user表中所有记录
Cursor cursor = resolver.query(uri, null, null, null, "userid desc");
while(cursor.moveToNext()){
    //操作
}

//把id为1的记录的name字段值更改新为finch
ContentValues updateValues = new ContentValues();
updateValues.put("name", "finch");
Uri updateIdUri = ContentUris.withAppendedId(uri, 1);
resolver.update(updateIdUri, updateValues, null, null);

//删除id为2的记录
Uri deleteIdUri = ContentUris.withAppendedId(uri, 2);
resolver.delete(deleteIdUri, null, null);
```

5.ContentObserver

ContentObserver(内容观察者)，目的是观察特定Uri引起的数据库的变化，继而做一些相应的处理，它类似于数据库技术中的触发器(Trigger)，当ContentObserver所观察的Uri发生变化时，便会触发它.

下面是使用内容观察者监听短信的例子：

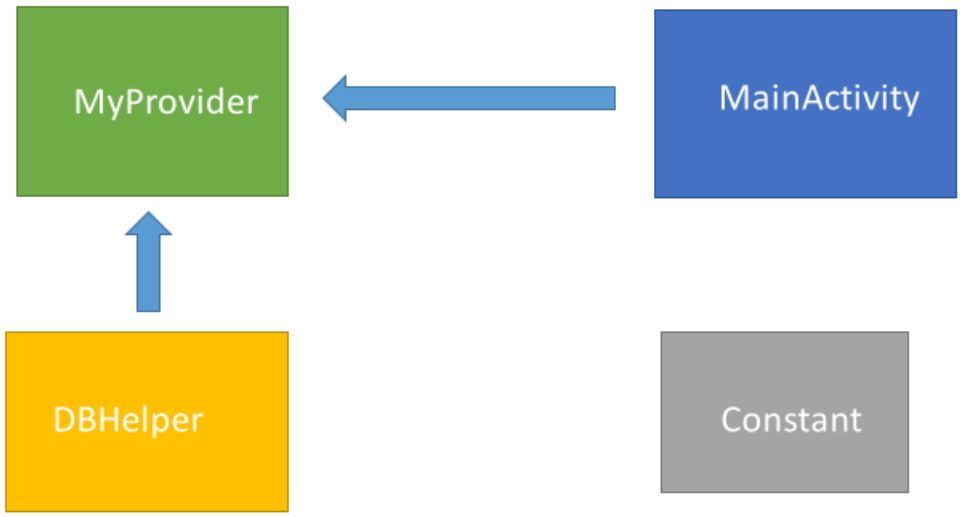
```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

//注册观察者Observser
this.getContentResolver().registerContentObserver(Uri.parse("content://sms"),true,new SMSObserver(new

    }

    private final class SMSObserver extends ContentObserver {

        public SMSObserver(Handler handler) {
            super(handler);

        }


        @Override
        public void onChange(boolean selfChange) {

 Cursor cursor = MainActivity.this.getContentResolver().query(
Uri.parse("content://sms/inbox"), null, null, null, null);

            while (cursor.moveToNext()) {
                StringBuilder sb = new StringBuilder();

                sb.append("address=").append(
                        cursor.getString(cursor.getColumnIndex("address")));

                sb.append(";subject=").append(
                        cursor.getString(cursor.getColumnIndex("subject")));

                sb.append(";body=").append(
                        cursor.getString(cursor.getColumnIndex("body")));

                sb.append(";time=").append(
                        cursor.getLong(cursor.getColumnIndex("date")));

                System.out.println("--------has Receivered SMS::" + sb.toString());


            }

        }

    }
}
```

同时可以在ContentProvider发生数据变化时调用 getContentResolver().notifyChange(uri, null)来通知注册在此URI上的访问者。

```
public class UserContentProvider extends ContentProvider {
   public Uri insert(Uri uri, ContentValues values) {
      db.insert("user", "userid", values);
      getContext().getContentResolver().notifyChange(uri, null);
   }
}
```

6.实例说明

数据源是SQLite, 用ContentResolver操作ContentProvider。



Constant.java（储存一些常量）

```
public class Constant {

    public static final String TABLE_NAME = "user";

    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_NAME = "name";


    public static final String AUTOHORITY = "cn.scu.myprovider";
    public static final int ITEM = 1;
    public static final int ITEM_ID = 2;

    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/user";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/user";

    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTOHORITY + "/user");
}
```

DBHelper.java(操作数据库)

```
public class DBHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "finch.db";
    private static final int DATABASE_VERSION = 1;

    public DBHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db)  throws SQLException {
        //创建表格
        db.execSQL("CREATE TABLE IF NOT EXISTS "+ Constant.TABLE_NAME + "("+ Constant.COLUMN_ID +" IN
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)  throws SQLException {
        //删除并创建表格
        db.execSQL("DROP TABLE IF EXISTS "+ Constant.TABLE_NAME+";");
        onCreate(db);
    }
}
```

MyProvider.java(自定义的ContentProvider)

```
public class MyProvider extends ContentProvider {

    DBHelper mDbHelper = null;
    SQLiteDatabase db = null;
```

```
    private static final UriMatcher mMatcher;
    static{
        mMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        mMatcher.addURI(Constant.AUTOHORITY,Constant.TABLE_NAME, Constant.ITEM);
        mMatcher.addURI(Constant.AUTOHORITY, Constant.TABLE_NAME+"/#", Constant.ITEM_ID);
    }


    @Override
    public String getType(Uri uri) {
        switch (mMatcher.match(uri)) {
        case Constant.ITEM:
            return Constant.CONTENT_TYPE;
        case Constant.ITEM_ID:
            return Constant.CONTENT_ITEM_TYPE;
        default:
            throw new IllegalArgumentException("Unknown URI"+uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // TODO Auto-generated method stub
        long rowId;
        if(mMatcher.match(uri)!=Constant.ITEM){
            throw new IllegalArgumentException("Unknown URI"+uri);
        }
        rowId = db.insert(Constant.TABLE_NAME,null,values);
        if(rowId>0){
            Uri noteUri=ContentUris.withAppendedId(Constant.CONTENT_URI, rowId);
            getContext().getContentResolver().notifyChange(noteUri, null);
            return noteUri;
        }

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

    @Override
    public boolean onCreate() {
        // TODO Auto-generated method stub
        mDbHelper = new DBHelper(getContext());

        db = mDbHelper.getReadableDatabase();

        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        // TODO Auto-generated method stub
        Cursor c = null;
        switch (mMatcher.match(uri)) {
        case Constant.ITEM:
            c =  db.query(Constant.TABLE_NAME, projection, selection, selectionArgs, null, null, sort
            break;
        case Constant.ITEM_ID:
            c = db.query(Constant.TABLE_NAME, projection,Constant.COLUMN_ID + "="+uri.getLastPathSegm
            break;
        default:
            throw new IllegalArgumentException("Unknown URI"+uri);
        }

        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        // TODO Auto-generated method stub
        return 0;
    }

}
```

MainActivity.java(ContentResolver操作)

```
public class MainActivity extends Activity {
    private ContentResolver mContentResolver = null;
    private Cursor cursor = null;
         @Override
        protected void onCreate(Bundle savedInstanceState) {
            // TODO Auto-generated method stub
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

               TextView tv = (TextView) findViewById(R.id.tv);

                mContentResolver = getContentResolver();
                tv.setText("添加初始数据 ");
                for (int i = 0; i < 10; i++) {
                    ContentValues values = new ContentValues();
                    values.put(Constant.COLUMN_NAME, "fanrunqi"+i);
                    mContentResolver.insert(Constant.CONTENT_URI, values);
                }

                tv.setText("查询数据 ");
                cursor = mContentResolver.query(Constant.CONTENT_URI, new String[]{Constant.COLUMN_ID
                if (cursor.moveToFirst()) {
                    String s = cursor.getString(cursor.getColumnIndex(Constant.COLUMN_NAME));
                    tv.setText("第一个数据：  "+s);
                }
        }

}
```

最后在manifest申明

```
<provider android:name="MyProvider" android:authorities="cn.scu.myprovider" />
```

本文中代码下载