第八章 Content Provider

北京邮电大学 计算机学院
刘伟
w.liu@foxmail.com

8.1 ContentProvider

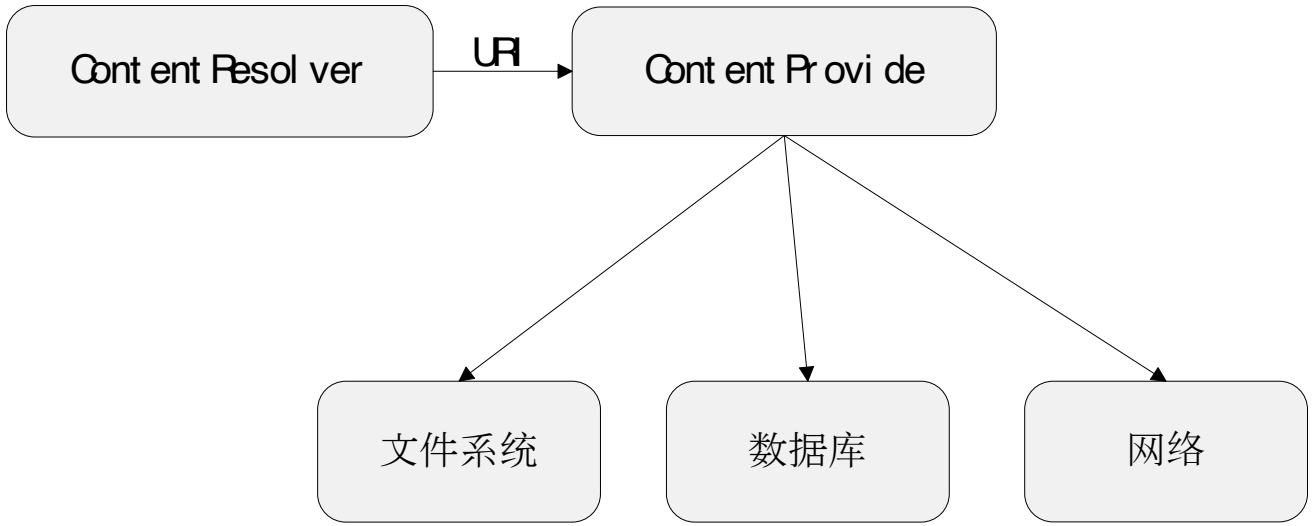
- Android应用程序运行在不同的进程空间中，因此不同应用程序的数据是不能够直接访问的
- 为了增强程序之间的数据共享能力，Android系统提供了像SharedPreferences这类类简单的跨越程序边界的访问方法，但这些方法都存在一定的局限性
- ContentProvider（数据提供者）是应用程序之间共享数据的一种接口机制，是一种更为高级的数据共享方法，可以指定需要共享的数据，而其它应用程序则则可在不知道数据来源、路径的情况下，对共享数据进行查询、添加、删除和更新等操作

8.1 ContentProvider

- 在Android系统中，许多Android系统内置的数据也是通过ContentProvider提供给用户使用，例如通讯录、音视频文件和图像文件等
- 在创建ContentProvider前，首先要实现底层的数据源，数据源包括数据库、文件系统或网络等，然后继承ContentProvider类中实现基本数据操作的接口函数，包括添加、删除、查找和更新等功能

8.1 ContentProvider

- 调用者不能直接调用ContentProvider的接口函数，而需要使用ContentResolver对象，通过URI间接调用ContentProvider，调用关系如下图所示:



8.1 ContentProvider

- 在ContentResolver对象与ContentProvider进行交互时，通过URI确定要访问的ContentProvider数据集
- 在发起一个请求的过程中，Android系统根据URI确定处理这个查询的ContentProvider，然后初始化ContentProvider所有需要的资源，这个初始化的工作是Android系统完成的，无需程序开发人员参与
- 一般情况下只有一个ContentProvider对象，但却可以同时与多个ContentResolver进行交互

8.1 ContentProvider

- ContentProvider完全屏蔽了数据提供组件的数据存储方法
- 在程序开发人员看来，数据提供者通过ContentProvider提供了一组标准的数据操作接口，但却无需知道数据提供者的内部数据的存储方法
- 数据提供者可以使用SQLite数据库存储数据，也可以通过文件系统或SharedPreferences存储数据，甚至是使用网络存储的方法，这些数据的存储方法和存储设备对数据使用者都是不可见的
- 同时，也正是这种屏蔽模式，很大程度上简化的ContentProvider的使用方法，使用者只要调用ContentProvider提供的接口函数，即可完成所有的数据操作，而数据存储方法则是ContentProvider设计者需要考虑的问题

8.1 ContentProvider

- ContentProvider的数据集类似于数据库的数据表，每行是一条记录，每列具有相同的数据类型
- 如下表所示。每条记录都包含一个长型的字段_ID，用来唯一标识每条记录
- ContentProvider可以提供多个数据集，调用者使用URI对不同数据集的数据进行操作
- ContentProvider数据集

_ID	NAME	AGE	HEIGHT
1	Tom	21	1.81
2	Jim	22	1.78

8.1 ContentProvider

- URI是通用资源标志符（Uniform Resource Identifier），用来定位远程或本地的可用资源
- ContentProvider使用的URI语法结构如下:
- content://<authority>/<data_path>/<id>
  - content://是通用前缀，表示该URI用于ContentProvider定位资源，无需修改
  - <authority>是授权者名称，用来确定具体由哪一个ContentProvider提供资源
  - 因此，一般<authority>都由类的小写全称组成，以保证唯一性。<data_path>是数据路径，用来确定请求的是哪个数据集

8.1 ContentProvider

- 如果ContentProvider仅提供一个数据集，数据路径则是可以省略的
- 但如果ContentProvider提供多个数据集，数据路径则必须指明具体是哪一个数据集
  - 数据集的数据路径可以写成多段格式，例如people/girl和/people/boy。<id>是数据编号，用来唯一确定数据集中的一条记录，用来匹配数据集中_ID字段的值
- 如果请求的数据并不只限于一条数据，则<id>是可以省略，例如
  - 请求整个people数据集的URI应写为

```
content://edu.bupt.peopleprovider/people
```

  - 请求people数据集中第3条数据的URI则应写为

```
content://edu.bupt.peopleprovider/people/3
```

8.2 创建数据提供者

- 程序开发人员通过继承ContentProvider类可以创建一个新的数据提供者，过程可以分为三步
  - 继承ContentProvider，并重载六个函数
  - 声明CONTENT_URI，实现UriMatcher
  - 注册ContentProvider

8.2 创建数据提供者

- 继承ContentProvider，并重载六个函数
  - 新建立的类继承ContentProvider后，共有六个函数需要重载，分别是
    - delete():删除数据集
    - insert()：添加数据集
    - qurey()：查询数据集
    - update()：更新数据集
    - onCreate()：初始化底层数据集和建立数据连接等工作
    - getType()：返回指定URI的MIME数据类型
      - 如果URI是单条数据，则返回的MIME数据类型应以vnd.android.cursor.item开头
      - 如果URI是多条数据，则返回的MIME数据类型应以vnd.android.cursor.dir/开头

8.2 创建数据提供者

- 继承ContentProvider，并重载六个函数
  - 新建立的类继承ContentProvider后，IDE会提示程序开发人员需要重载部分的代码，并自动生成需要重载的代码框架
  - 下面的代码是IDE自动生成的代码框架

```
1    import android.content.*;
2    import android.database.Cursor;
3    import android.net.Uri;
4
5    public class PeopleProvider extends ContentProvider{
6
7            @Override
8            public int delete(Uri uri, String selection, String[]
     selectionArgs) {
9          // TODO Auto-generated method stub
10         return 0;
```

8.2 创建数据提供者

- 继承ContentProvider，并重载六个函数

```
11          }
12
13          @Override
14          public String getType(Uri uri) {
15        // TODO Auto-generated method stub
16        return null;
17          }
18
19          @Override
20          public Uri insert(Uri uri, ContentValues values) {
21        // TODO Auto-generated method stub
22        return null;
23          }
24
25          @Override
26          public boolean onCreate() {
27        // TODO Auto-generated method stub
```

8.2 创建数据提供者

- 继承ContentProvider，并重载六个函数

```
28          return false;
29              }
30
31              @Override
32              public Cursor query(Uri uri, String[] projection, String
    selection,
33                  String[] selectionArgs, String sortOrder) {
34          // TODO Auto-generated method stub
35          return null;
36              }
37
38              @Override
39              public int update(Uri uri, ContentValues values, String
    selection,
40                  String[] selectionArgs) {
41          // TODO Auto-generated method stub
42          return 0;
43              }
44      }
```

8.2 创建数据提供者

- 声明CONTENT_URI，实现UriMatcher
  - 在新构造的ContentProvider类中，经常需要判断URI是单条数据还是多条数据，最简单的方法是构造一个UriMatcher
  - 为了便于判断和使用URI，一般将URI的授权者名称和数据路径等内容声明为静态常量，并声明CONTENT_URI

8.2 创建数据提供者

- 声明CONTENT_URI和构造UriMatcher的代码如下：

```
1    public static final String AUTHORITY = "edu.bupt.peopleprovider";
2    public static final String PATH_SINGLE = "people/#";
3    public static final String PATH_MULTIPLE = "people";
4    public static final String CONTENT_URI_STRING = "content://" + AUTHORITY
     + "/" + PATH_MULTIPLE;
5    public static final Uri  CONTENT_URI = Uri.parse(CONTENT_URI_STRING);
6    private static final int MULTIPLE_PEOPLE = 1;
7    private static final int SINGLE_PEOPLE = 2;
8
9    private static final UriMatcher uriMatcher;
10   static {
11    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
12    uriMatcher.addURI(AUTHORITY, PATH_SINGLE, MULTIPLE_PEOPLE);
13    uriMatcher.addURI(AUTHORITY, PATH_MULTIPLE, SINGLE_PEOPLE);
14   }
```

8.2 创建数据提供者

- 声明CONTENT_URI，实现UriMatcher
  - 第1行声明了URI的授权者名称
  - 第2行声明了单条数据的数据路径
  - 第3行声明了多条数据的数据路径
  - 第4行声明了CONTENT_URI的字符串形式
  - 第5行则正式声明了CONTENT_URI
  - 第6行声明了多条数据的返回代码
  - 第7行声明了单条数据的返回代码
  - 第9行声明了UriMatcher
  - 第10行到第13行的静态构造函数中，声明了UriMatcher的匹配方式和返回代码

8.2 创建数据提供者

- 声明CONTENT_URI，实现UriMatcher
  - 在第11行UriMatcher的构造函数中，UriMatcher.NO_MATCH是URI无匹配时的返回代码
  - 第12行的addURI()函数用来添加新的匹配项，语法如下：

```
1    public void  addURI  (String authority, String path, int code)
```

  - authority表示匹配的授权者名称
  - path表示数据路径
  - #可以代表任何数字
  - code表示返回代码

8.2 创建数据提供者

- 声明CONTENT_URI，实现UriMatcher
  - 使用UriMatcher时，则可以直接调用match()函数，对指定的URI进行判断，示例代码如下：

```
1    switch(uriMatcher.match(uri)){
2     case MULTIPLE_PEOPLE:
3             //多条数据的处理过程
4             break;
5     case SINGLE_PEOPLE:
6             //单条数据的处理过程
7             break;
8     default:
9             throw new IllegalArgumentException("不支持的URI:" + uri);
10   }
```

8.2 创建数据提供者

- 注册ContentProvider
  - 在完成ContentProvider类的代码实现后，需要在AndroidManifest.xml文件中进行注册
  - 注册ContentProvider使用<provider>标签，示例代码如下:

```
1    <application android:icon="@drawable/icon"
     android:label="@string/app_name">
2            <provider android:name = ".PeopleProvider"
3             android:authorities = "edu.bupt.peopleprovider"/>
4    </application>
```

  - 在上面的代码中，注册了一个授权者名称为edu.bupt.peopleprovider的ContentProvider，其实现类是PeopleProvider

8.3 使用数据提供者

- 使用ContentProvider并不需要直接调用类中的数据操作函数，而是通过Android组件都具有的ContentResolver对象，通过URI进行数据操作
- 程序开发人员只需要知道URI和数据集的数据格式，则可以进行数据操作，解决不同应用程序之间的数据共享问题
- 每个Android组件都具有一个ContentResolver对象，获取ContentResolver对象的方法是调用getContentResolver()函数

```
3    ContentResolver resolver = getContentResolver();
```

8.3 使用数据提供者

- 查询操作
  - 在获取到ContentResolver对象后，程序开发人员则可以使用query()函数查询目标数据
  - 下面的代码是查询ID为2的数据

```
String KEY_ID = "_id";
String KEY_NAME = "name";
String KEY_AGE = "age";
String KEY_HEIGHT = "height";

Uri uri = Uri.parse(CONTENT_URI_STRING + "/" + "2";
Cursor cursor = resolver.query(uri,
 new String[] {KEY_ID, KEY_NAME, KEY_AGE, KEY_HEIGHT}, null, null,
null);
```

8.3 使用数据提供者

- 查询操作
  - 从上面的代码不难看出，在URI中定义了需要查询数据的ID后，在query()函数中则没有必要再加入其他的查询条件
  - 如果需要获取数据集中的全部数据，则可直接使用CONTENT_URI，此时ContentProvider在分析URI时将认为需要返回全部数据
- ContentResolver的query()函数与SQLite数据库的query()函数非常相似，语法结构如下：

```
1  Cursor query(Uri uri, String[] projection, String selection, String[]
   selectionArgs, String sortOrder)
```

  - uri定义了查询的数据集，projection定义了从应返回数据的哪些属性，selection定义了返回数据的查询条件

8.3 使用数据提供者

- 添加操作
  - 向ContentProvider中添加数据有两种方法
    - 一种是使用insert()函数，向ContentProvider中添加一条数据
    - 另一种是使用bultInsert()函数，批量的添加数据
  - 下面的代码说明了如何使用insert()函数添加单条数据

```
1    ContentValues values = new ContentValues();
2    values.put(KEY_NAME, "Tom");
3    values.put(KEY_AGE, 21);
4    values.put(KEY_HEIGHT, 1.81f);
5
6    Uri newUri = resolver.insert(CONTENT_URI, values);
```

8.3 使用数据提供者

- 添加操作
  - 下面的代码说明了如何使用bultInsert()函数添加多条数据

```
1    ContentValues[] arrayValues = new ContentValues[10];
2    //实例化每一个ContentValues
3    int count = resolver.bultInsert(CONTENT_URI, arrayValues);
```

8.3 使用数据提供者

- 删除操作
  - 删除操作需要使用delete()函数
  - 如果需要删除单条数据，则可以在URI中指定需要删除数据的ID
  - 如果需要删除多条数据，则可以在selection中声明删除条件
  - 下面代码说明了如何删除ID为2的数据

```
1  Uri uri = Uri.parse(CONTENT_URI_STRING + "/" + "2");
2  int result = resolver.delete(uri, null, null);
```

  - 也可以在selection将删除条件定义为ID大于4的数据

```
1  String selection = KEY_ID + ">4";
2  int result = resolver.delete(CONTENT_URI, selection, null);
```

8.3 使用数据提供者

- 更新操作
  - 更新操作需要使用update()函数，参数定义与delete()函数相同，同样可以在URI中指定需要更新数据的ID，也可以在selection中声明更新条件
  - 下面代码说明了如何更新ID为7的数据

```
1    ContentValues values = new ContentValues();
7    values.put(KEY_NAME, "Tom");
8    values.put(KEY_AGE, 21);
9    values.put(KEY_HEIGHT, 1.81f);
2
3    Uri uri = Uri.parse(CONTENT_URI_STRING + "/" + "7");
4    int result = resolver.update(uri, values, null, null);
```

习题：

1. 分别使用手动建库和代码建库的方式，创建名为test.db数据库，并建立staff数据表，表内的属性值如下表所示：

属性	数据类型	说明
_id	integer	主键
name	text	姓名
sex	text	性别
department	text	所在部门
salary	float	工资

习题：

2.利用第1题所建立的数据库和staff表，为程序提供添加、删除和更新等功能，并尝试将下表中的数据添加到staff表中。

_id	name	sex	department	salary
1	Tom	male	computer	5400
2	Einstein	male	computer	4800
3	Lily	female	1.68	5000
4	Warner	male		
5	Napoleon	male		

3.建立一个ContentProvider，用来共享第2题所建立的数据库。