第十四章. Content Provider的使用

`android`

学习目标：

- 内容提供者的基本概念。
- Uri的概念，ContentUris与UriMatcher的常用方法
- 扩展系统的内容提供者将数据暴露出去

课程内容：

概述

Content Provider内部如何保存数据由其设计者决定 .但所有的Content Provider都实现了一组通的方法来提供数据的增删改查功能.
客户端通常不会直接使用这些方法,大多数据是通过ContentResolver对象实现对Content Provider的操作.开发人员可以通过调用 Activity或者其它应用程序组伯的实现类中的getContentResolver()方法来获得Content Provider对象 如
ContentResolver cr=getContentResolver();
使用ContentResolver提供的方法可以获得Content Provider中任何感兴趣的数据.
当开始查询时,Android系统确认查询的目标Content Provider并确何它正在运行.系统会初始化所有的Content Provider类的对象,开发人员不必完寄居此娄操作,实际上,开发人员根本不会直接使用Content Provider,通常,每个类型的Content Provider仅有一个单独的实例,但是该实例能与位于不同应用程序和进程的多个ContentResolver类对象通信.不同进程之间通信由Content Provider类和ContentResolver类处理.

数据模型

Content Provider使用基本数据库模型的简单表格来提供其中的数据,这里每行代表一条记录,每列代表特定类型和含义的数据.例如,联系人的信息可以如下

```
_ID     NAME    NUMBER      EMAIL
001     张三    123******   123******@163.com
002     李四    132******   132******@google.com
003     王五    414******   414******@qq.com
```

每条记录包含一个数值型的_ID字段,它用于在表格中唯一标记该记录.ID能用于匹配相关表格中的记录,例如在一个表格中查询联系人电话,在另一个表格中查询照片
查询返回一个Cursor对象,它能遍历各行各列来读取各个字段的值.对于各个类型的数据,Cursor对象都提供专用的方法.因此,为了读取字段的数据,开发人员必须知道当前字段包含的数据类型

URI的用法

在电脑术语中，统一资源标识符（Uniform Resource Identifier，或URI)是一个用于标识某一互联网资源名称的字符串。 该种标识允许用户对任何（包括本地和互联网）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。

每个Content Provider提供公共的URI来唯一标识其数据集,管理多个数据集(多个表格)的Content Provider为每个都提供了单独的URI.所提供的URI都以content://作为前缀”content://”模式表示数据由Content Provider来管理
如果自定义Content Provider,则应该也是其URI定义一个常量,以简化客户端代码并让日后更新更加简洁.Android为当前平台提供的Content Provider定义了CONTENT_URI常量.匹配电话号码到联系人表格的URI和匹配保存联系人照片表格的URI分别如下.
android.provider.Contacts.Phones.CONTENT.URI;
android.provider.Contacts.Photos.CONTENT.URI;

URI常量用于所有与Content Provider的交互中.每个ContentResolver方法使用URI作为其第一个参数.它标识ContentResolver应该使用哪个provider以及其中的哪个表格.
下面是Content URI重要部分的总结
content://com.hua.employeeprovider/dba/001
上面的URI分四部分
第一部分:标准的前缀,用于标识该数据由Content Provider管理,它永远不用修改
第二部分:URI的authority部分,它标识该Content Provider.对于第三方应用,该部分应该是完整的类名(小写)来保证唯一性,在元素的authorities属性中声明authority.
第三部分: Content Provider的路径部分,用于决定哪类数据被请求,如果Content Provider仅提供一种数据类型,这部分可以没有.如果提供几种类型,包括子类型,这部分可以由几部分组成.
第四部分:被请求的特定的记录的ID值.这是被请求记录的_ID值.如果请求不仅限于单条记录该部分及前面的斜线应该删除

系统中预定义Content Provider

Android系统为常用数据类型提供了很多预定义的Content Provider(声音,视频,图片,联系人),它他大部分位于android.provider包中,开发人员可以查询这些provider经获得其中包含的信息.android系统提供的常见Content Provider说明如下

```
1,Browser:读取或修改书签,浏览历史或网络搜索.
2,CallLog:查看或更新通话历史.
3,Contacts:获取,修改或保存联系人
4,LiveFolders:由Content Provider提供 内容的特定文件夹
5,MediaStrore:访问声音,视频和图片
6,Setting:查看和获取蓝牙设置,铃声和其它设备偏好.
7,SearchRecentSuggestions:该类能为应用程序创建简单的查询建议提供者,它基于
```

```
近期查询提供建议.
    8,SyncStateContract:用于使用数据数组帐号关联数据的Content Provider约束,希望使用标准方式保存数据的provider可以使用它.
    9,UserDictionary:在可以预测文本输入时,提供用户定义的单词给输入法使用.应用程序和输入法能增加到该字典,单词能关联频率信息和本地化信息.
```

查询数据

在Content Provider中查询数据,开发人员需要知道以下信息:
标识该Content Provider的URI
需要查询的数据字段名称
字段中数据的类型
如果需要查询特定记录,那么还需要知道该记录的ID值.
ContentResolver.query()或Activity.managed-Query()方法都可以完成查询功能,这两个方法使用的参数相同,都返回Cursor对象.其区别在于managedQuery()方法记Activity来管理Cursor生命周期,而query()方法则需要程序员自己管理.query()方法的声明如下

```
Public final Cursor query(Uri uri,String[] projection,String selection,String[]
selectionArgs,String sortOrder)
    Uri:用于查询的Content Providerr URI值
    Projection:由需要查询 的列名组成的数组,如果为NULL则不胜枚举查询全部列
    Selection:类似SQL中的WHERE子句,用于增加条件来完成数据过虑
    SelectionArgs:用于替换selection中可以使用?表示的变量值
    sortOrder:类似SQL中的ORDER BY 子句,用于实现排序功能
    返回值:Cursor对象,它位于第一条记录之前,或者为NULL
```

为了限制仅返回一条记录,可以在URL结尾加该记录的_ID值,即将匹配ID值的字符串作为URI路径部分的结尾片段.

自定义Content Provider

如果开发人员希望共享自己的数据,可以自定义Content Provider
步骤如下

- 建立数据存储系统,大多数Content Provider使用Android文件存储方法或者SQLite数据库

保存数据,但是开发人员可以使用任何方式存储.

- 继承ContentProvider类来提供数据访问方式
- 在应用程序的AndroidManifest文件中声明ContentProvider

UriMatcher

UriMatcher 类主要用于匹配Uri。
使用方法如下。
首先第一步，初始化：

```
UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
```

第二步注册需要的Uri:

```
matcher.addURI("com.whunf", "people", PEOPLE);
matcher.addURI("com.whunf", "person/#", PEOPLE_ID);
```

第三部，与已经注册的Uri进行匹配:

```
Uri uri = Uri.parse("content://" + "com.whunf" + "/people");
int match = matcher.match(uri);
        switch (match)
        {
            case PEOPLE:
                return "vnd.android.cursor.dir/people";
            case PEOPLE_ID:
                return "vnd.android.cursor.item/people";
            default:
                return null;
        }
```

match方法匹配后会返回一个匹配码Code，即在使用注册方法addURI时传入的第三个参数。
上述方法会返回"vnd.android.cursor.dir/person".
总结:

常量 UriMatcher.NO_MATCH 表示不匹配任何路径的返回码

`#` 号为通配符

`*` 号为任意字符

官方SDK说明中关于Uri的注册是这样写的：

```
private static final UriMatcher sURIMatcher = new
UriMatcher(UriMatcher.NO_MATCH);
    static
    {
        sURIMatcher.addURI("contacts", "/people", PEOPLE);
        sURIMatcher.addURI("contacts", "/people/#", PEOPLE_ID);
        sURIMatcher.addURI("contacts", "/people/#/phones", PEOPLE_PHONE
S);
        sURIMatcher.addURI("contacts", "/people/#/phones/#", PEOPLE_PHO
NES_ID);
        sURIMatcher.addURI("contacts", "/people/#/contact_methods", PEO
PLE_CONTACTMETHODS);
        sURIMatcher.addURI("contacts", "/people/#/contact_methods/#", P
EOPLE_CONTACTMETHODS_ID);
        sURIMatcher.addURI("contacts", "/deleted_people", DELETED_PEOPL
E);
        sURIMatcher.addURI("contacts", "/phones", PHONES);
        sURIMatcher.addURI("contacts", "/phones/filter/*", PHONES_FILTE
R);
        sURIMatcher.addURI("contacts", "/phones/#", PHONES_ID);
        sURIMatcher.addURI("contacts", "/contact_methods", CONTACTMETHO
DS);
        sURIMatcher.addURI("contacts", "/contact_methods/#", CONTACTMET
HODS_ID);
        sURIMatcher.addURI("call_log", "/calls", CALLS);
        sURIMatcher.addURI("call_log", "/calls/filter/*", CALLS_FILTER)
;
        sURIMatcher.addURI("call_log", "/calls/#", CALLS_ID);
    }
```

ContentUris

ContentUris 类用于获取Uri路径后面的ID部分

- 为路径加上ID: withAppendedId(uri, id)

比如有这样一个Uri

```
1.  Uri uri = Uri.parse("content://com.whunf/people")
```

通过withAppendedId方法，为该Uri加上ID

```
1.  Uri resultUri = ContentUris.withAppendedId(uri, 10);
```

最后resultUri为: content://com.whunf/people/10

- 从路径中获取ID: parseId(uri)

```
1.  Uri uri = Uri.parse("content://com.whunf/people/10")
2.  long personid = ContentUris.parseId(uri);
```

最后personid 为 :10

作业

在上一章作业中添加ContentProvider，并且实现从另一个工程中读取已存储的数据显示。