ContentProvider

deepwaterooo

2021 年 12 月 15 日

目录

1 Content Provider 内存承载器

- ContentProvider 的底层是采用 Android 中的 Binder 机制

1.1 统一资源标识符（URI）

- 定义：Uniform Resource Identifier，即统一资源标识符
- 作用：唯一标识 ContentProvider & 其中的数据
  - 外界进程通过 URI 找到对应的 ContentProvider & 其中的数据，再进行数据操作
- 具体使用
  - URI 分为系统预置 & 自定义，分别对应系统内置的数据（如通讯录、日程表等等）和自定义数据库
    - 关于系统预置 URI 此处不作过多讲解，需要的同学可自行查看
    - 此处主要讲解自定义 URI


```
Uri uri = Uri.parse("content://com.carson.provider/User/1") // 设置 URI
// 上述 URI 指向的资源是：名为 `com.carson.provider`的 `ContentProvider` 中表名 为 `User` 中的 `id`为 1 的数据

// 特别注意：URI 模式存在匹配通配符：* # (两个)
//  *：匹配任意长度的任何有效字符的字符串
//     以下的 URI 表示 匹配 provider 的任何内容
//     content://com.example.app.provider/*
// #：匹配任意长度的数字字符的字符串
//     以下的 URI 表示 匹配 provider 中的 table 表的所有行
//     content://com.example.app.provider/table/#
```

- uri 的各个部分在安卓中都是可以通过代码获取的，下面我们就以下面这个 uri 为例来说下获取各个部分的方法：
- http://www.baidu.com:8080/wenku/jiatiao.html?id=123456&name=jack

```
getScheme() // 获取 Uri 中的 scheme 字符串部分，在这里是 http
getHost()   // 获取 Authority 中的 Host 字符串，即 www.baidu.com
getPost()   // 获取 Authority 中的 Port 字符串，即 8080
getPath()   // 获取 Uri 中 path 部分，即 wenku/jiatiao.html
getQuery()  // 获取 Uri 中的 query 部分，即 id=15&name=du
```

1.2 MIME 数据类型

- 作用：指定某个扩展名的文件用某种应用程序来打开
  - 如指定.html 文件采用 text 应用程序打开、指定.pdf 文件采用 flash 应用程序打开

1.2.1 ContentProvider 根据 URI 返回 MIME 类型

```
ContentProvider.geType(uri) ;
```

1.2.2 MIME 类型组成

- 每种 MIME 类型由 2 部分组成 = 类型 + 子类型
- MIME 类型是一个包含 2 部分的字符串

```
text / html
// 类型 = text、子类型 = html
text/css
text/xml
application/pdf
```

1.2.3 MIME 类型形式

- MIME 类型有 2 种形式：单条记录, 或是多条记录

```
// 形式 1：单条记录
vnd.android.cursor.item/自定义
// 形式 2：多条记录（集合）
vnd.android.cursor.dir/自定义

// 注：
  // 1. vnd：表示父类型和子类型具有非标准的、特定的形式。
  // 2. 父类型已固定好（即不能更改），只能区别是单条还是多条记录
  // 3. 子类型可自定义
```

- 实例说明

```
<-- 单条记录 -->
  // 单个记录的 MIME 类型
  vnd.android.cursor.item/vnd.yourcompanyname.contenttype

  // 若一个 Uri 如下
  content://com.example.transportationprovider/trains/122
  // 则 ContentProvider 会通过 ContentProvider.geType(url) 返回以下 MIME 类型
  vnd.android.cursor.item/vnd.example.rail

<-- 多条记录 -->
  // 多个记录的 MIME 类型
  vnd.android.cursor.dir/vnd.yourcompanyname.contenttype
  // 若一个 Uri 如下
  content://com.example.transportationprovider/trains
  // 则 ContentProvider 会通过 ContentProvider.geType(url) 返回以下 MIME 类型
  vnd.android.cursor.dir/vnd.example.rail
```

1.3 ContentProvider 类

1.3.1 组织数据方式

- ContentProvider 主要以表格的形式组织数据
  - 同时也支持文件数据，只是表格形式用得比较多
  - 每个表格中包含多张表，每张表包含行 & 列，分别对应记录 & 字段, 同数据库

1.3.2 主要方法

- 进程间共享数据的本质是：添加、删除、获取 & 修改（更新）数据 (CRUD: create / read / update / delete)
- 所以 ContentProvider 的核心方法也主要是上述几个作用

```
  // 外部进程向 ContentProvider 中添加数据
  public Uri insert(Uri uri, ContentValues values)

  // 外部进程 删除 ContentProvider 中的数据
  public int delete(Uri uri, String selection, String[] selectionArgs)

  // 外部进程更新 ContentProvider 中的数据
  public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)

  // 外部应用 获取 ContentProvider 中的数据
  public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs,  String sortOrder)

// 注：
  // 1. 上述 4 个方法由外部进程回调，并运行在 ContentProvider 进程的 Binder 线程池中（不是主线程）
  // 2. 存在多线程并发访问，需要实现线程同步
```

```
    // a. 若 ContentProvider 的数据存储方式是使用 SQLite & 并且只有一个，则不需要，因为 SQLite 内部实现好了线程同步，若是多个 SQL
    // b. 若 ContentProvider 的数据存储方式是内存，则需要自己实现线程同步

// 2 个其他方法 -->
// ContentProvider 创建后 或 打开系统后其它进程第一次访问该 ContentProvider 时 由系统进行调用
public boolean onCreate()
// 注：运行在 ContentProvider 进程的主线程，故不能做耗时操作

// 得到数据类型，即返回当前 Url 所代表数据的 MIME 类型
public String getType(Uri uri)
```

- Android 为常见的数据（如通讯录、日程表等）提供了内置了默认的 ContentProvider
- 但也可根据需求自定义 ContentProvider，但上述 6 个方法必须重写
- 数据访问的方法 insert，delete 和 update 可能被多个线程同时调用，此时必须是线程安全的。(前面提到过)
- 如果操作的数据属于集合类型，那么 MIME 类型字符串应该以 vnd.android.cursor.dir/ 开头，
  - 要得到所有 tablename 记录:Uri 为 content://com.wang.provider.myprovider/tablename，那么返回的 MIME 类型字符串应该为：vnd.android.cursor.dir/table。
- 如果要操作的数据属于非集合类型数据,那么 MIME 类型字符串应该以 vnd.android.cursor.item/ 开头，
  - 要得到 id 为 10 的 tablename 记录,Uri 为 content://com.wang.provider.myprovider/tablename 那么返回的 MIME 类型字符串为：vnd.android.cursor.item/tablename 。
- ContentProvider 类并不会直接与外部进程交互，而是通过 ContentResolver 类

1.4 ContentResolver 类

- 统一管理不同 ContentProvider 间的操作
  - 即通过 URI 即可操作不同的 ContentProvider 中的数据
  - 外部进程通过 ContentResolver 类从而与 ContentProvider 类进行交互
- 为什么要使用通过 ContentResolver 类从而与 ContentProvider 类进行交互，而不直接访问 ContentProvider 类?
  - 一般来说，一款应用要使用多个 ContentProvider，若需要了解每个 ContentProvider 的不同实现从而再完成数据交互，操作成本高 & 难度大
  - 所以再 ContentProvider 类上加多了一个 ContentResolver 类对所有的 ContentProvider 进行统一管理。
- ContentResolver 类提供了与 ContentProvider 类相同名字 & 作用的 4 个方法

```
// 外部进程向 ContentProvider 中添加数据
public Uri insert(Uri uri, ContentValues values)

// 外部进程 删除 ContentProvider 中的数据
public int delete(Uri uri, String selection, String[] selectionArgs)

// 外部进程更新 ContentProvider 中的数据
public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)

// 外部应用 获取 ContentProvider 中的数据
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
```

- 实例说明

```
// 使用 ContentResolver 前，需要先获取 ContentResolver
// 可通过在所有继承 Context 的类中 通过调用 getContentResolver() 来获得 ContentResolver
ContentResolver resolver =  getContentResolver();

// 设置 ContentProvider 的 URI
Uri uri = Uri.parse("content://cn.scu.myprovider/user");

// 根据 URI 操作 ContentProvider 中的数据
// 此处是获取 ContentProvider 中 user 表的所有记录
Cursor cursor = resolver.query(uri, null, null, null, "userid desc");
```

- Android 提供了 3 个用于辅助 ContentProvider 的工具类：
  - ContentUris
  - UriMatcher
  - ContentObserver

1.5 ContentUris 类

- 作用：操作 URI
- 核心方法有两个：withAppendedId（）&parseId（）

```
// withAppendedId() 作用：向 URI 追加一个 id
Uri uri = Uri.parse("content://cn.scu.myprovider/user")
Uri resultUri = ContentUris.withAppendedId(uri, 7);
// 最终生成后的 Uri 为：content://cn.scu.myprovider/user/7

// parseId() 作用：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
```

1.6 UriMatcher 类

- 在 ContentProvider 中注册 URI
- 根据 URI 匹配 ContentProvider 中对应的数据表

```
// 步骤 1：初始化 UriMatcher 对象
    UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
    // 常量 UriMatcher.NO_MATCH = 不匹配任何路径的返回码
    // 即初始化时不匹配任何东西

// 步骤 2：在 ContentProvider 中注册 URI(addURI())
    int URI_CODE_a = 1;
    int URI_CODE_b = 2;
    matcher.addURI("cn.scu.myprovider", "user1", URI_CODE_a);
    matcher.addURI("cn.scu.myprovider", "user2", URI_CODE_b);
    // 若 URI 资源路径 = content://cn.scu.myprovider/user1 ，则返回注册码 URI_CODE_a
    // 若 URI 资源路径 = content://cn.scu.myprovider/user2 ，则返回注册码 URI_CODE_b

    //如果 match() 方法匹配 content://com.wang.provider.myprovider/tablename/11 路径，返回匹配码为 2
    matcher.addURI("com.wang.provider.myprovider", "tablename/#", 2);

// 步骤 3：根据 URI 匹配 URI_CODE，从而匹配 ContentProvider 中相应的资源（match()）
@Override public String getType(Uri uri) {
      Uri uri = Uri.parse(" content://cn.scu.myprovider/user1");

      switch(matcher.match(uri)) {
      // 根据 URI 匹配的返回码是 URI_CODE_a
```

```
        // 即 matcher.match(uri) == URI_CODE_a
        case URI_CODE_a:
            // 如果根据 URI 匹配的返回码是 URI_CODE_a，则返回 ContentProvider 中的名为 tableNameUser1 的表
            return tableNameUser1;
        case URI_CODE_b:
            // 如果根据 URI 匹配的返回码是 URI_CODE_b，则返回 ContentProvider 中的名为 tableNameUser2 的表
            return tableNameUser2;
    }
}
```

- 注意，添加第三个个 URI 时，路径后面的 id 采用了通配符形式“#”，表示只要前面三个部分都匹配上了就 OK。
- 第三步，注册完需要匹配的 Uri 后，可以使用 matcher.match(Uri) 方法对输入的 Uri 进行匹配，如果匹配就返回对应的匹配码，匹配码为调用 addURI() 方法时传入的第三个参数。

1.7 ContentObserver 类

- 定义：内容观察者
- 作用：观察 Uri 引起 ContentProvider 中的数据变化 & 通知外界（即访问该数据访问者）
  - 当 ContentProvider 中的数据发生变化（增、删 & 改）时，就会触发该 ContentObserver 类

```
// 步骤 1：注册内容观察者 ContentObserver
// 通过 ContentResolver 类进行注册，并指定需要观察的 URI
getContentResolver().registerContentObserver(uri);

// 步骤 2：当该 URI 的 ContentProvider 数据发生变化时，通知外界（即访问该 ContentProvider 数据的访问者）
public class UserContentProvider extends ContentProvider {
    public Uri insert(Uri uri, ContentValues values) {
        db.insert("user", "userid", values);
        // 通知访问者
        getContext().getContentResolver().notifyChange(uri, null);
    }
}
// 步骤 3：解除观察者
getContentResolver().unregisterContentObserver(uri);
// 同样需要通过 ContentResolver 类进行解除
```

- 上面说得可能还不是太彻底，下面再重新写一下
- 如果 ContentProvider 的访问者需要知道数据发生的变化，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。只给出类中监听部分的代码：

```
public class MyProvider extends ContentProvider {
   public Uri insert(Uri uri, ContentValues values) {
      db.insert("tablename", "tablenameid", values);
      getContext().getContentResolver().notifyChange(uri, null);
   }
}
```

- 而访问者必须使用 ContentObserver 对数据（数据采用 uri 描述）进行监听，当监听到数据变化通知时，系统就会调用 ContentObserver 的 onChange() 方法：

```
getContentResolver().registerContentObserver(Uri.parse("content://com.ljq.providers.personprovider/person"),
       true, new PersonObserver(new Handler()));
public class PersonObserver extends ContentObserver{
   public PersonObserver(Handler handler) {
      super(handler);
```

```
    }
    public void onChange(boolean selfChange) {
        //to do something
    }
}
```

1.8 优点

1.8.1 安全

- ContentProvider 为应用间的数据交互提供了一个安全的环境：允许把自己的应用数据根据需求开放给其他应用进行增、删、改、查，而不用担心因为直接开放数据库权限而带来的安全问题

1.8.2 访问简单 & 高效

- 对比于其他对外共享数据的方式，数据访问方式会因数据存储的方式而不同：
  - 采用文件方式对外共享数据，需要进行文件操作读写数据；
  - 采用 Sharedpreferences 共享数据，需要使用 sharedpreferences API 读写数据, 这使得访问数据变得复杂 & 难度大。
  - 而采用 ContentProvider 方式，其解耦了底层数据的存储方式，使得无论底层数据存储采用何种方式，外界对数据的访问方式都是统一的，这使得访问简单 & 高效
  - 如一开始数据存储方式采用 SQLite 数据库，后来把数据库换成 MongoDB，也不会对上层数据 ContentProvider 使用代码产生影响