Content Providers Personalizados

(contenidos elaborados por academiaandroid.com)

Introducción

En este tutorial explicaremos cómo compartir información entre diferentes aplicaciones, utilizando un mecanismo proporcionado por Android llamado Content Provider. En el se tratarán los siguientes puntos:

- Pasos para crear un Content Provider personalizado.
- Uso de un Content Provider personalizado.
- Métodos que implementa la clase Content Provider.

1. Pasos para crear un Content Provider personalizado

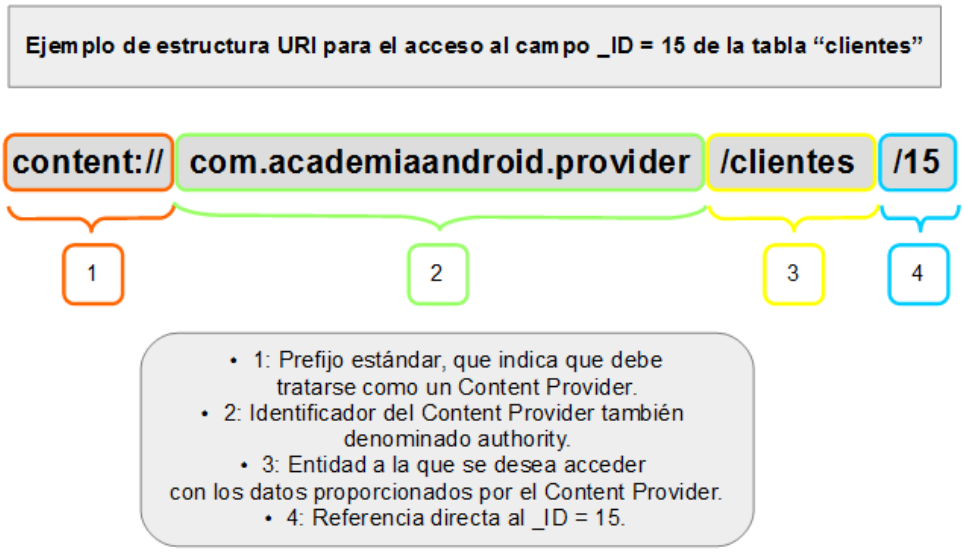
Android nos ofrece la posibilidad de crear nuestros propios proveedores de contenidos, y para ello nos proporciona el package `android.content`, dónde se encuentra la clase `ContentProvider`, que será la encargada de almacenar todos los datos a los que se accederá desde cualquier aplicación.

Referencia a Content Provider a través de una URI:

A cada elemento Content Provider se le asocia una URI única, que lo representa y a través de la cual el resto de componentes de otras aplicaciones acceden a él.

> **URI** (Uniform Resource Identificator). El denominado "identificador de recursos uniforme" está formado por una cadena de caracteres que identifica de manera inequívoca los recursos de una red.

En el siguiente diagrama vemos la **referencia a un Content Provider a través de una URI**:



Implementación de Content Provider personalizado:

A continuación, vamos a ver los pasos necesarios para la creación de un Content Provider propio:

1. Se debe definir una clase que herede de la clase base ContentProvider:

```
public class InformacionComun extends ContentProvider {
```

2. Se declara e inicializa una constante de la clase Uri, que recogerá la URI (identificador de recursos uniforme, formado por una cadena de caracteres que identifica de manera inequívoca los recursos de una red) que identificará de manera única el Content Provider (revisar imagen con ejemplo de estructura URI):

```
public static final Uri CONTENT_URI = Uri.parse("content://com.academiaandroid.provider/clientes");
```

3. Indicar el tipo de almacenamiento que se va a establecer. Este almacenamiento puede ser desde una base de datos SQLite a un archivo local.

4. Declarar e inicializar un array de Strings, con las columnas que se desean obtener (ejemplo para base de datos SQLte):

```
public static String[] columnas = new String[]{"_id",
                                               "cliente",
                                               "telefono",
                                               "descripcion"};
```

5. Se sobrescribirán los métodos que implementa la clase **ContentProvider**:

```
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    return 0;
}

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

@Override
public Uri insert(Uri uri, ContentValues values) {
    return null;
}

@Override
public boolean onCreate() {
    return false;
}

@Override
public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder) {
    return null;
}

@Override
public int update(Uri arg0, ContentValues arg1, String arg2, String[] arg3)
{
    return 0;
}
```

6. Por último, dentro del AndroidManifest.xml, y más concretamente dentro de la etiqueta `<application>`, se añadirá el provider, como se muestra a continuación (el atributo `exported` se establece en true para habilitar

el envío de mensajes de fuentes externas a la aplicación):

```
 <application>
[...]
    <provider
            android:name="com.academiaandroid.provider.InformacionComun"
            android:authorities="com.academiaandroid.provider.InformacionComun"
            android:exported="true">
    <provider/>
[...]

 </application>
```

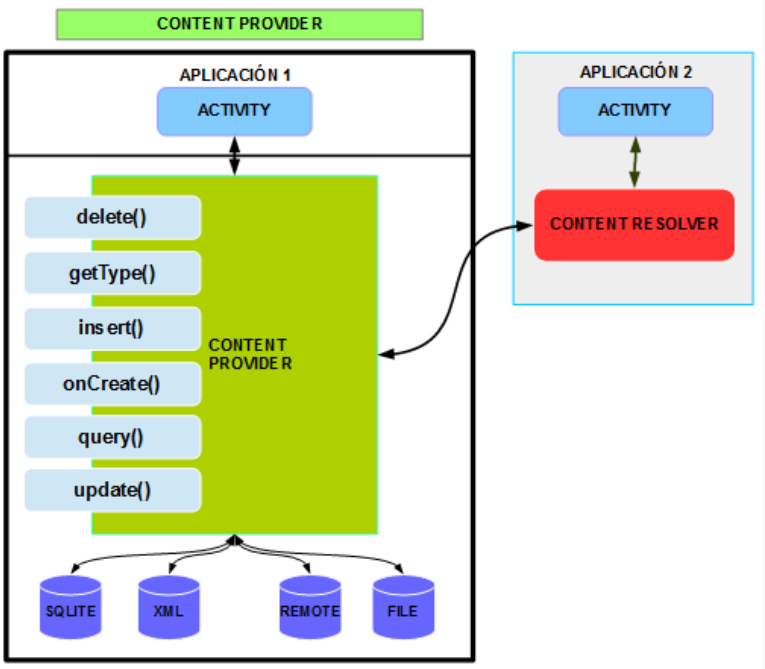
2. Uso de un Content Provider personalizado

En este punto, se describirán los pasos necesarios para usar un Content Provider personalizado utilizando la clase `ContentResolver`.

En primer lugar se definirá el concepto de **ContentResolver**, que es el mecanismo que proporciona Android para acceder a todas las acciones sobre el Content Provider. Un ContentResolver nos permite acceder a los datos del Content Provider, ya que es el encargado de recibir las solicitudes del cliente, ejecutar la acción solicitada y mostrar los resultados de dicha ejecución.

Lógica de intercambio de datos entre aplicaciones

En el siguiente diagrama se representa esquemáticamente los componentes y elementos que intervienen en este intercambio de datos: base de datos, métodos, Content Provider, Content Resolver, Activities y Aplicaciones.



Implementación de un ContentResolver

1. En primer lugar, debemos disponer de un Content Provider para acceder a los datos que contiene.

2. Posteriormente, dentro de nuestra segunda aplicación (será la encargada de realizar las solicitudes al Content Provider definido), se declara e inicializa la clase ContentResolver, referenciándole el método getContentResolver(), que nos permitirán acceder a los métodos sobrescritos en la clase que hereda de la clase base ContentProvider:

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

3. Además será necesario declarar e inicializar la clase **Cursor** (clase que permite almacenar el resultado de una consulta realizada a una base de datos), referenciándole el objeto ContentResolver, e invocando los métodos que implementa la clase base ContentProvider. A continuación, se muestra un ejemplo de consulta de datos en SQLite, que recibe como argumentos la URI del Content Provider, un array de Strings con las columnas a consultar, el filtro de las filas a devolver (cláusula SQL WHERE), valor a comprobar dentro de la cláusula WHERE, y por último el orden de salida de los resultados devueltos:

   Construcción del método query():

```
public final Cursor query (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder
```

   Ejemplo:

```
Cursor c = resolver.query(InformacionComun.CONTENT_URI, InformacionComun.columnas, null, null, null);
```

4. Finalmente, se recorrerán los resultados almacenados de la consulta en el objeto Cursor, comprobando en cada iteración que existe el registro siguiente:

```
if(c.moveToFirst())
{
    do
    {
        Log.i("ContentResolver",c.getString(0));
    }while(c.moveToNext());
}
```

3. Métodos que implementa la clase ContentProvider.

Para finalizar las bases teóricas de la implementación de un Content Provider personalizado, revisamos los métodos que se sobrescribían de la clase ContentProvider, para realizar tareas básicas de creación, lectura, actualización y eliminación de datos (CRUD).

- El método `delete()` permite manejar solicitudes de eliminación de datos, recibiendo entre sus parámetros la URI del Content Provider, y los argumentos de selección y orden definidos. Devuelve el número de filas afectadas:

```
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        return 0;
    }
```

- El método `getType()` se utiliza para identificar el tipo de dato que devuelve el Content Provider. Estos datos se expresan como MIME Type (estándar que especifica cómo debe un programa enviar archivos multimedia, informando del tipo y subtipo del contenido del archivo):

```
    @Override
    public String getType(Uri uri) {
```

```
        return null;
    }
```

- El método `insert()` proporciona la funcionalidad de insertar datos, devolviendo la URI que hace referencia al registro introducido:

```
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        return null;
    }
```

- El método `onCreate()` es dónde se inicializa el tipo de almacenamiento utilizado para el Content Provider. Un ejemplo muy común sería inicializar la clase que herede de la clase base SQLiteOpenHelper:

```
    @Override
    public boolean onCreate() {
        return false;
    }
```

- A través del método `query()`, podemos consultar los datos almacenados. Este método devuelve un objeto Cursor con los datos solicitados al Content Provider:

```
    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        return null;
    }
```

- El método `update()` permitirá modificar y actualizar los datos solicitados. Este método devuelve el número de registros afectados:

```
    @Override
    public int update(Uri uri, ContentValues arg1, String arg2, String[] arg3)
    {
        return 0;
    }
```

Bibliografía:

- Documentación oficial Android clase **ContentProvider**: http://developer.android.com/reference/android/content/ContentProvider.html
- Documentación oficial Android clase **ContentResolver**: http://developer.android.com/reference/android/content/ContentResolver.html
- Documentación oficial Android clase **Cursor**: http://developer.android.com/reference/android/database/Cursor.html
- Documentación oficial Android clase **ArrayList**: http://developer.android.com/reference/java/util/ArrayList.html
- Fundamentos básicos **Content Provider**: http://developer.android.com/guide/topics/providers/content-provider-basics.html