Content Provider

Proveedor de Contenidos

Curso 2013 - 2014
2 º DAM
Javier López Gutiérrez

Índice

Content Provider

Content Provider se trata de una superclase que nos proporciona Android para la gestión de datos entre diversas aplicaciones. A lo que se refiere esto es que mediante una base de datos (por norma general en android SQLite) o incluso un componente algo más sencillo como puede ser un ArrayList, este componente nos proporciona datos a diversas aplicaciones. Es muy utilizado para algunas aplicaciones como puede ser la agenda, mensajería, llamadas, etc. Todas estas aplicaciones utilizan un mismo Content Provider que les suministra los datos de tus contactos.

Por norma general, una de las aplicaciones será la encargada de gestionar los datos del Content Provider y el resto que quieran utilizar los recursos de esta superclase solamente tendrán derechos de lectura sobre los datos, de esta manera se tiene un mayor control sobre la gestión de los datos de nuestra base de datos, además de que se podrían llegar a dar conflictos si dos o más aplicaciones accedieran a un mismo recurso y lo modificaran y guardaran, de esta manera nos ahorramos esos posibles problemas.

Tipos de Content Provider

Por decirlo de alguna manera, existen dos tipos de Content Provider, los ya existente que han sido creados por los desarrolladores del sistema Android, y los otros Providers son los que crearemos y gestionaremos nosotros mismos, aunque realmente ambos son la misma clase, lo que unos ya vienen definidos con sus métodos y sus constantes y los otros debemos nosotros ser quienes creen la clase y redefinan los métodos requeridos por dicha clase.

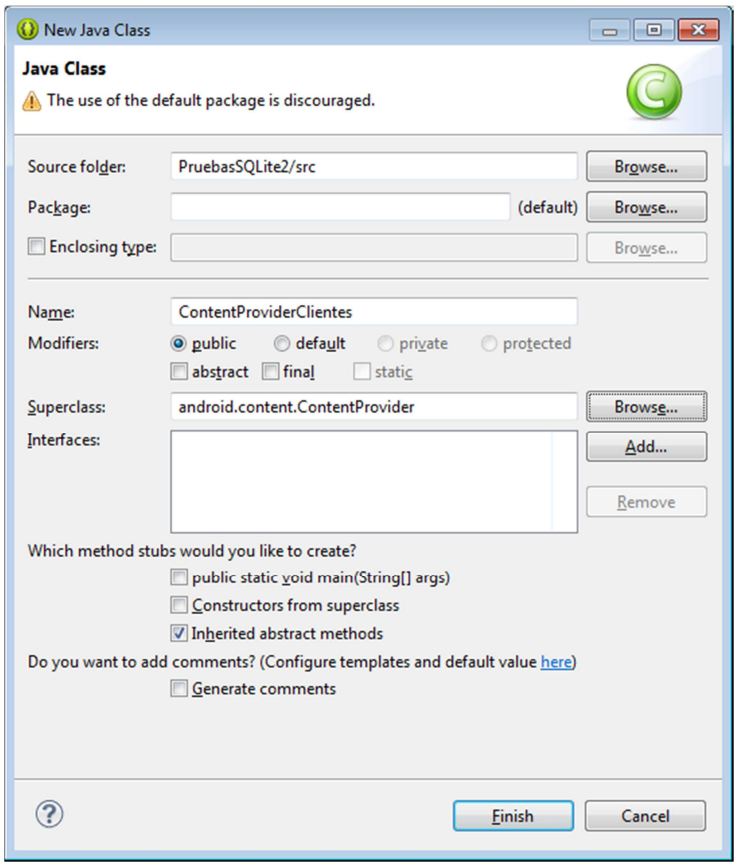
Definición de un Content Provider

Para definir un Content Provider necesitaremos dar dos pasos:

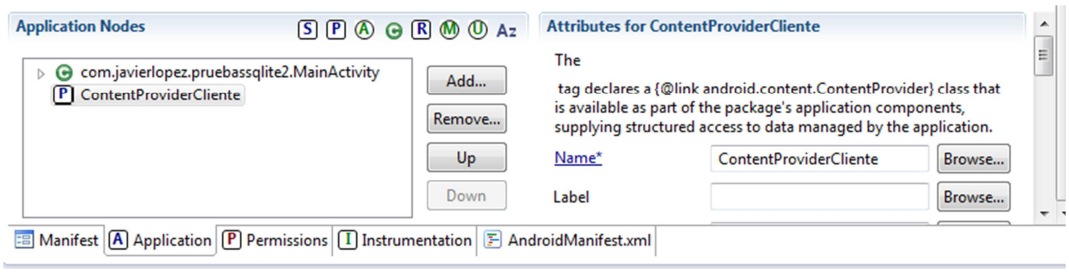
1. El primero será el de crear una clase nueva que extienda de Content Provider, podemos generarla con el asistente de nuestro programa de edición o una vez creada una clase genérica colocar a la derecha del nombre de la clase ‘extends Content Provider’ como se puede ver en la imagen que va a continuación.

```
public class ContentProviderCliente extends ContentProvider {

    // Definición del CONTENT_URI
    private static final String uri = "content://com.javierlopez.pruebassqlite2/productos";

    public static final Uri CONTENT_URI = Uri.parse(uri);
```

Con el asistente sería más sencillo ya que al generar la clase nueva ya generaría todas los métodos que debemos sobrescribir posteriormente, de todos modos, más adelante mencionare dichos métodos.



2. Una vez tenemos creada nuestra clase debemos declararla y para ello debemos ir al fichero 'AndroidManifest.xml'. Una vez estemos en el fichero debemos seleccionar la pestaña de la parte inferior que pone 'Aplication' y añadimos un nuevo 'Provider'. Una vez que lo tenemos añadido le seleccionamos y le damos el nombre que queramos y un poco más abajo habrá que rellenar el atributo que se llama 'Authorities'. Este atributo es el que se encarga de enlazar nuestro Provider a nuestra superclase asique es de vital importancia poner bien la ruta, esta ruta es la misma ruta del 'package' que se define al generar nuestro proyecto en este paso sería 'com.javierlopez.pruebassqlite2'.



Con estos dos pasos importantes ya tendriamos definido nuestro Content Provider pero nada mas, ya que aun no esta creada ni la base de datos ni tampoco estan redefinidos los metodos que gestionan el Content Provider.

Creación base de datos SQLite

Aunque es un tema que se abordara en otro proyecto, el Content Provider necesita de un lugar de almacenamiento y de distribución de datos. No profundizare mucho en este tema y solamente diré como crear la base de datos y como añadirle unos registros para que posteriormente se pueda trabajar con el Provider.

Para crear nuestra base de datos, del mismo modo que con el Content Provider, necesitaremos crear una nueva clase que extienda de la superclase 'SQliteOpenHelper'. En principio solo necesitaremos de esta superclase para nuestra gestión. Una vez generada la clase añadiremos el código para crear las tablas oportunas y rellenarlas con datos.

```
package com.javierlopez.pruebassqlite2;

import android.content.Context;

public class SQLiteBase extends SQLiteOpenHelper {

    //Sentencia que generara posteriormente la tabla con cada campo y su clave primaria
    String sqlCrear = "CREATE TABLE Productos "
            + "(_id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + " nombre TEXT, "
            + " precio TEXT )";

    /**
     *
     * @param context contexto en el que se ejecutara nuestra base de datos
     * @param name nombre con el que se guardara en el terminal la base
     * @param factory
     * @param version de la base para saber cuantas modificaciones ha tenido o en que estado esta
     */
    public SQLiteBase(Context context, String name, CursorFactory factory,
            int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        //Se ejecuta la sentencia creada anteriormente y se genera nuestra tabla
        db.execSQL(sqlCrear);

        // Insertamos 15 productos de ejemplo
        for (int i = 1; i <= 15; i++) {
            // Generamos los datos de muestra
            String nombre = "Precio" + i;
            String precio = i + "." + i;

            // Insertamos los datos en la tabla Productos
            db.execSQL("INSERT INTO Productos (nombre, precio) "
                    + "VALUES ('" + nombre + "', '" + precio + "')");
        }
```

```
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int versionAnterior,
            int versionNueva) {

        //Se borra la version anterior de la tabla
        db.execSQL("DROP TABLE IF EXISTS Productos");

        // Se crea la nueva versión de la tabla
        db.execSQL(sqlCrear);

    }
```

Una vez tenemos este código en nuestra clase que creara la base de datos con los datos de prueba será momento de codificar la otra clase que habíamos definido del Content Provider que será la que gestione la base de datos.

Configuración Content Provider

Lo primero que haremos para poder conectarnos a nuestro Content Provider y posteriormente poder gestionar la base de datos será crear nuestra URI que es una cadena de texto que funciona a grandes rasgos como las líneas de texto de un navegador web.

La dirección URI consta de tres componentes a destacar:

1. Como prefijo en nuestra URI deberemos colocar 'content://', este prefijo lo que hace es que el programa interprete que tiene que ejecutar dicha ruta como un Content Provider.

2. El segundo tramo de nuestra cadena será la dirección literal de nuestro proyecto, para saber dónde está alojado el Content Provider. En mi caso esta ruta es 'com.javierlopez.pruebassqlite2'.

3. Como tercer y último argumento hemos de colocar el nombre de la entidad a la que queremos acceder, en nuestro caso como es una tabla llamada 'Productos' debemos colocar '/productos' para que el programa interprete a donde tiene que ir y que recurso ha de conseguir por si hubiera más de uno.

Un ejemplo de URI podría ser este: 'content://com.javierlopez.pruebassqlite2/productos'. También sería posible acceder directamente a un registro en concreto de nuestra base de datos mediante la URI añadiendo posteriormente al nombre de la tabla el número de registro al que queremos acceder. Para este propósito tenemos la clase proporcionada por Android llamada 'UriMatcher' que nos facilitara la tarea a la hora de trabajar con estas rutas y acceder a los recursos de la base de datos.

Para ello en nuestro código debemos crear una variable del tipo 'UriMatcher' y crear dos constantes para definir si queremos acceder a la tabla de un modo genérico o si por el contrario queremos acceder mediante el identificador de cada producto.

```
// Definición del CONTENT_URI
private static final String uri = "content://com.javierlopez.pruebassqlite2/productos";

public static final Uri CONTENT_URI = Uri.parse(uri);

// Necesario para UriMatcher
private static final int PRODUCTOS = 1;
private static final int PRODUCTOS_ID = 2;
private static final UriMatcher uriMatcher;
```

```
// Inicializamos el UriMatcher
static {
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI("com.javierlopez.pruebassqlite2", "productos",
            PRODUCTOS);
    uriMatcher.addURI("com.javierlopez.pruebassqlite2",
            "productos/#", PRODUCTOS_ID);
}
```

Por ultimo definiremos y crearemos la propia base de datos en sí, para ellos crearemos una constante con el nombre que queramos y crearemos nuestra clase extendida de 'SQLiteOpenHelper'. Para facilitar el código podemos crear constantes para indicar el nombre de la base y la versión, aunque no sería estrictamente necesario. Es importante que como primer parámetro se le pase el contexto, al no estar en la activista principal debemos obtenerlo y para ello usaremos el método 'getContext()'.

Una vez definida nuestra URI, el UriMatcher encargado de gestionar las consultas y creada la base de datos ya tendríamos los recursos necesarios creados para que nuestro Content Provider funcione.

El siguiente paso será la codificación de los métodos para gestionar en si la base de datos. Esta parte es complicada y ardua, aunque una vez entendido el funcionamiento puede resultar sencilla.

```
// Clase interna para declarar las constantes de columna
public static final class Productos implements BaseColumns {
    private Productos() {
    }

    // Nombres de columnas
    public static final String COL_NOMBRE = "nombre";
    public static final String COL_PRECIO = "precio";
}

// Base de datos
private SQLiteBase clidbh;
private static final String BD_NOMBRE = "DBProductos";
private static final int BD_VERSION = 1;
private static final String TABLA_PRODUCTOS = "Productos";
```

Los métodos que debemos sobrescribir (recordar que si creamos la clase Content Provider y luego le hicimos el extend debemos crear uno a uno estos métodos, de la otra forma ya se pre generaran automáticamente) son:

- onCreate();
- query();
- insert();
- update();
- delete();
- getType();

OnCreate()

En el primer método (onCreate();), como en todos los proyectos de android, será el método que se ejecutara según se llame a la clase, en este método iniciaremos las variables necesarias (todas las variables que hemos definido anteriormente URI, UriMatcher, la base de datos, etc). Este método se ejecutara una vez al ser llamado por el programa que quiera acceder al Content Provider que hemos definido.

```
@Override
public boolean onCreate() {

    clidbh = new SQLiteBase(getContext(), BD_NOMBRE, null, BD_VERSION);

    return true;
}
```

Query()

Una vez definido 'onCreate()', el siguiente método será 'query()'. Este método se encarga de gestionar el cursor que nos indicara que registro hemos seleccionado y devolvernos los datos seleccionados, la funciona es la misma que los cursores de bases de datos, pero en este caso tenemos este método que será el que vaya a la base de datos y llame a los datos o registros que requerimos. Este método requiere de cinco parámetros:

1. el primero es el Uri, de esta manera sabemos si queremos acceder a la tabla o por el contrario queremos acceder a un id concreto.
2. El segundo es una lista de nombres de las columnas que vamos a mirar.
3. El tercer parámetro que se le pasa es un criterio de selección para saber si posteriormente ha de añadir un where al final para buscar por el identificador.
4. Como cuarto parámetro se le pasa una lista de variables que utilizara el criterio anterior.
5. Por último se le pasa como parámetro un criterio de ordenación para si hubiese que mostrarlo ordenado de algún modo.

Una vez que se le pasa el 'uri' en el código de 'query()' se comprueba que tipo de 'uri' ha llegado mediante el método '.matcher()' este método nos dirá si la 'id' de 'uri' es 1 o 2 y de esa

manera devolver todos los registros encontrados según la verificación o solamente el del id seleccionado.

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

    // Si es una consulta a un ID concreto construimos el WHERE
    String where = selection;
    if (uriMatcher.match(uri) == PRODUCTOS_ID) {
        where = "_id=" + uri.getLastPathSegment();//.getLastPathSegment() devuelve el ultimo segmento
        //de la cadena uri de esa manera solamente coge el valor del id del registro
    }

    //Se abre la base de datos para hacer la llamada a la tabla.
    SQLiteDatabase db = clidbh.getWritableDatabase();

    //Se obtiene el cursor con el dato o los datos que se hayan requerido
    Cursor c = db.query(TABLA_PRODUCTOS, projection, where, selectionArgs,
            null, null, sortOrder);

    //Se devuelve el cursor para que el programa principal se encargue de gestionarlo
    return c;
}
```

Update() y delete()

Los métodos 'update()' y 'delete()' son similares al 'query()', la diferencia que existe es que en update y delete se le pasan 3 parámetros, el primero común para todos los métodos que vamos a ver que es el 'uri'. Otro parámetro es el criterio de selección mencionado anteriormente, el siguiente son las variables que usara el criterio de selección. Además para update habrá que pasarle los nuevos valores del registro que vamos a modificar. El código quedaría de este modo.

```
@Override
public int update(Uri uri, ContentValues values, String selection,
        String[] selectionArgs) {

    int cont;

    // Si es una consulta a un ID concreto construimos el WHERE
    String where = selection;
    if (uriMatcher.match(uri) == PRODUCTOS_ID) {
        where = "_id=" + uri.getLastPathSegment();
    }

    SQLiteDatabase db = clidbh.getWritableDatabase();

    cont = db.update(TABLA_PRODUCTOS, values, where, selectionArgs);

    return cont;
}
```

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

    int cont;


    SQLiteDatabase db = clidbh.getWritableDatabase();

    cont = db.delete(TABLA_PRODUCTOS, selection, selectionArgs);

    return cont;

}
```

Insert()

El método de inserción de datos, 'insert()', es distinto a los anteriores, aunque no difiere en mucho. Este método nos retorna la 'uri' del último elemento añadido en nuestra base de datos. Como datos de entrada para este método solamente necesitaremos del 'uri' de la tabla y de un 'ContentValues' con los valores del nuevo registro. Una vez dentro del método lo que haremos en nuestro caso será, crear de nuevo un objeto 'SQLiteDataBase' para gestionar nuestra entrada de datos y como en el caso del 'query()' usaremos el método '.getWritableDatabase()' para escribir los datos. Una vez hemos definido nuestra clase de gestión proporcionada por Android.

```
@Override
public Uri insert(Uri uri, ContentValues values) {

    long regId = 1;

    //Gestor de la base de datos
    SQLiteDatabase db = clidbh.getWritableDatabase();
    regId = db.insert(TABLA_PRODUCTOS, null, values);

    //Se añade nuestro nuevo registro a la tabla
    Uri newUri = ContentUris.withAppendedId(CONTENT_URI, regId);

    //Se devuelve el valor del uri
    return newUri;
}
```

Una vez generada la sentencia llamando al método '.insert()' lo que se hace es adjuntar este nuevo registro al final de nuestra tabla.

GetType()

En principio con los otros métodos podríamos trabajar sin ningún problema, este método realmente lo que hace es identificar el tipo de dato que devuelve el 'content provider'. Este método es importante para que Android identifique el tipo de datos que devuelve el 'content provider' y ver que aplicaciones son capaces de procesar dichos datos.

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

    //Se genera un match de la uri que se le pasa
    int match = uriMatcher.match(uri);

    //Se comprueba la id mediante un switch para ver que tipo de dato es
    //si se trata de una dirección o si se trata de un unico registro
    switch (match) {
    case PRODUCTOS:
        return "vnd.android.cursor.dir/vnd.javierlopez.productos";
    case PRODUCTOS_ID:
        return "vnd.android.cursor.item/vnd.javierlopez.productos";
    default:
        return null;
    }
}
```

Una vez definidos estos métodos ya tendríamos el 'Content Provider' configurado y listo para funcionar en nuestra aplicación.

Bibliografía

Datos referentes a definición de que es un 'Content Provider' con sus métodos a redefinir:

- http://developer.android.com/reference/android/content/ContentProvider.html
- http://www.sgoliver.net/blog/?p=2057

Código de programa referente a la configuración de un 'Content Provider'. (El código general lo saque del primer 'url' el código del proyecto, aunque en el resto de links recogí código para que funcionara correctamente.)

- https://github.com/sgolivernet/curso-android-src/tree/master/android-content-providers-1
- http://www.sgoliver.net/blog/?p=2057
- http://stackoverflow.com/questions/5267348/calling-delete-method-in-custom-content-provider
- http://developer.android.com/guide/topics/providers/content-provider-basics.html