Что такое ContentProvider?

**ContentProvider** — механизм передачи данных от одного приложения к другому. Мы с вами уже сталкивались с ним несколькими уроками ранее, когда передавали ссылку на файл.

Как вы помните, в Android каждое приложение находится в "песочнице", то есть изолировано от других приложений. Соответственно, другие приложения не могут получить доступ к их данным.

А что, если приложение хранит список контактов пользователя, и другому приложению потребовалось получить доступ к этим данным? Как раз для таких случаев и придуман механизм ContentProvider.

В нашем случае реализация ContentProvider для доступа к БД избыточна (мы не предполагаем предоставления доступа для других приложений), но будет хорошим примером реализации.

Ничего сложного в его реализации нет — это всего лишь класс, в котором мы должны переопределить следующие методы:

* onCreate() — тут мы создаём БД
* query() — этот метод вызывается, когда кто-то запрашивает данные из БД
* insert() — вставляем данные в БД
* delete() — удаляем данные из БД
* update() — обновляем данные в БД

Доступ к данным в ContentProvider получается при помощи **URI**, например такого: content://com.skillberg.notes.provider/notes/1.

* content:// — схема. Она сообщает, что мы хотим получить доступ к контенту.
* com.skillberg.notes.provider — так называемый **Authority**, "адрес" ContentProvider.
* notes — путь к хранилищу данных определённого типа. Чтобы было понятнее — в большинстве случаев это просто имя таблицы, в которой хранятся данные.
* 1 — идентификатор контента. Например, значение поля \_id в таблице.

Давайте же приступим к реализации!

Создание ContentProvider

Разбираемся с URI

В первую очередь, добавьте две константы в класс NotesContract:

public final class NotesContract {

public static final String DB\_NAME = "notes.db";

public static final int DB\_VERSION = 1;

public static final String AUTHORITY = "com.skillberg.notes.provider

public static final String URI = "content://" + AUTHORITY;

// ...

Первая — **authority** ContentProvider.

Вторая — базовый **URI** для доступа к контенту.

Теперь в класс Notes внутри NotesContract добавьте константу, в которой будет храниться URI для доступа к конкретному типу данных — к заметкам:

public static abstract class Notes implements BaseColumns {

public static final String TABLE\_NAME = "notes";

public static final Uri URI = Uri.parse(NotesContract.URI + "/" + TABLE\_NAME);

// ...

И в него же добавьте константы, обозначающие типы данных:

// Список заметок

public static final String URI\_TYPE\_NOTE\_DIR = "vnd.android.cursor.dir/vnd.skillberg.note";

// Одна заметка

public static final String URI\_TYPE\_NOTE\_ITEM = "vnd.android.cursor.item/vnd.skillberg.note";

Теперь в пакете db создайте новый класс NotesProvider:

public class NotesProvider extends ContentProvider {

private NotesDbHelper notesDbHelper;

}

Переопределите в нём метод onCreate():

@Override

public boolean onCreate() {

notesDbHelper = new NotesDbHelper(getContext());

return true;

}

При обращении к ContentProvider последний должен понимать, о каком типе данных идёт речь. Например, если мы запросим список заметок, то он должен вернуть список, а если конкретную заметку — то одну заметку. Это разные типы данных.

Как он поймёт, о каком типе данных идёт речь? По **URI** — в нашем случае для списка заметок это будет content://com.skillberg.notes.provider/notes, а для конкретной заметки, скажем, с id = 1 — content://com.skillberg.notes.provider/notes/1.

В этом ему поможет специальный класс UriMatcher. Он принимает на вход **URI** и возвращает числовую константу, соответствующую определённому типу данных.

Сначала нужно задать **URI** и константы типов данных.

Добавьте две константы в NotesProvider:

private static final int NOTES = 1;

private static final int NOTE = 2;

Первая — список заметок, вторая — одна заметка.

Теперь добавьте в него же UriMatcher:

private static final UriMatcher URI\_MATCHER = new UriMatcher(UriMatcher.NO\_MATCH);

Единственный параметр конструктора — константа, которую нужно возвращать, если путь после **authority** не будет задан.

Инициализируем UriMatcher:

static {

URI\_MATCHER.addURI(NotesContract.AUTHORITY, "notes", NOTES);

URI\_MATCHER.addURI(NotesContract.AUTHORITY, "notes/#", NOTE);

}

Первый параметр — **authority**, второй — путь к данным, третий — константа, которую нужно вернуть, если **URI**совпадёт.

В первой строчке мы матчим список заметок. Во второй — конкретную заметку # — "плейсхолдер" для idзаметки.

Получился вот такой код:

public class NotesProvider extends ContentProvider {

private static final UriMatcher URI\_MATCHER = new UriMatcher(UriMatcher.NO\_MATCH);

private static final int NOTES = 1;

private static final int NOTE = 2;

static {

URI\_MATCHER.addURI(NotesContract.AUTHORITY, "notes", NOTES);

URI\_MATCHER.addURI(NotesContract.AUTHORITY, "notes/#", NOTE);

}

private NotesDbHelper notesDbHelper;

@Override

public boolean onCreate() {

notesDbHelper = new NotesDbHelper(getContext());

return true;

}

// ...

getType()

Пример места, в котором используется то, что мы сделали выше — метод getType().

Он принимает **URI** контента в качестве параметра и возвращает тип контента (не числовую константу, а тот, который мы определяли в контракте).

Давайте реализуем этот метод в нашем NotesProvider:

@Nullable

@Override

public String getType(@NonNull Uri uri) {

switch (URI\_MATCHER.match(uri)) {

case NOTES:

return NotesContract.Notes.URI\_TYPE\_NOTE\_DIR;

case NOTE:

return NotesContract.Notes.URI\_TYPE\_NOTE\_ITEM;

default:

return null;

}

}

Как видите, всё просто.

query()

ContentProvider.query() — метод, использующийся для получения данных, соответствующих определённым критериям.

Он принимает следующие параметры:

* Uri uri — **URI** контента.
* String[] projection — массив столбцов, которые нужно выбрать из таблицы. *Опциональный параметр.*
* String selection — параметры выборки (условие после WHERE в SQL). *Опциональный параметр.*
* String[] selectionArgs — аргументы выборки (будут экранированы). *Опциональный параметр.*
* String sortOrder — столбец, по которому сортировать данные и направление сортировки. *Опциональный параметр.*

Возвращает же он **Cursor**.

**Курсор** — специальный объект, позволяющий получать данные из БД "порциями", а не целиком.

Он будет особенно полезен, когда в базе тысячи или сотни тысяч записей (ведь для выборки всех данных за раз банально не хватит памяти), но на мобильных устройствах он нужен всегда из-за ограниченности ресурсов.

Итак, переопределите метод query():

@Nullable

@Override

public Cursor query(@NonNull Uri uri, @Nullable String[] projection, @Nullable String selection,

@Nullable String[] selectionArgs, @Nullable String sortOrder) {

}

В первую очередь в нём нужно получить инстанс базы данных:

**Обратите внимание:** поскольку мы будем использовать БД только для чтения, используем методgetReadableDatabase(). Метод getWritableDatabase() мы будем использовать только когда нам нужно будет что-то записать в базу.

Поскольку метод query() будет использоваться для доступа ко всей БД заметок, нужно понять, с каким именно типом данных работаем:

switch (URI\_MATCHER.match(uri)) {

case NOTES:

break;

case NOTE:

break;

default:

return null;

}

В случае, если нам нужен список заметок, нужно задать параметры сортировки (если они не заданы):

if (TextUtils.isEmpty(sortOrder)) {

sortOrder = NotesContract.Notes.COLUMN\_UPDATED\_TS + " DESC";

}

То есть, по умолчанию сортируем по убыванию столбца "последнее обновление".

Далее делаем выборку, банально "проксируя" запрос в БД, которую получили ранее:

return db.query(NotesContract.Notes.TABLE\_NAME,

projection,

selection,

selectionArgs,

null, // groupBy, не используется

null, // having, не используется

sortOrder);

В случае выборки одной заметки делаем то же самое, за исключением сортировки:

return db.query(NotesContract.Notes.TABLE\_NAME,

projection,

selection,

selectionArgs,

null,

null,

sortOrder);

В итоге, метод будет таким:

@Nullable

@Override

public Cursor query(@NonNull Uri uri, @Nullable String[] projection, @Nullable String selection,

@Nullable String[] selectionArgs, @Nullable String sortOrder) {

SQLiteDatabase db = notesDbHelper.getReadableDatabase();

switch (URI\_MATCHER.match(uri)) {

case NOTES:

if (TextUtils.isEmpty(sortOrder)) {

sortOrder = NotesContract.Notes.COLUMN\_UPDATED\_TS + " DESC";

}

return db.query(NotesContract.Notes.TABLE\_NAME,

projection,

selection,

selectionArgs,

null,

null,

sortOrder);

case NOTE:

String id = uri.getLastPathSegment();

if (TextUtils.isEmpty(selection)) {

selection = NotesContract.Notes.\_ID + " = ?";

selectionArgs = new String[]{id};

} else {

selection = selection + " AND " + NotesContract.Notes.\_ID + " = ?";

String[] newSelectionArgs = new String[selectionArgs.length + 1];

System.arraycopy(selectionArgs, 0, newSelectionArgs, 0, selectionArgs.length);

newSelectionArgs[newSelectionArgs.length - 1] = id;

selectionArgs = newSelectionArgs;

}

default:

return null;

}

}

В случае, если была запрошена конкретная заметка, мы получаем её идентификатор, используя метод getLastPathSegment().

После этого проверяем, не задан ли аргумент параметров выборки. Если не задан, то всё просто: создаём его и аргументы выборки.

Если же задан — дополняем его идентификатором, а так же копируем переданные ранее аргументы выборки и добавляем к ним полученный идентификатор.

insert()

Метод insert(), что логично, отвечает за вставку данных в БД.

Он принимает всего два параметра:

* Uri uri — URI, как и в query.
* ContentValues contentValues — Набор пар "ключ - значение" ("столбец - данные").

Метод возвращает **Uri** созданного элемента (если он был создан).

Переопределите его:

@Nullable

@Override

public Uri insert(@NonNull Uri uri, @Nullable ContentValues contentValues) {

}

Сначала получим инстанс БД, как и в query(), но пригодный для записи:

SQLiteDatabase db = notesDbHelper.getWritableDatabase();

Теперь нужно опять же понять, с каким типом данных мы работаем. В случае вставки мы используем URI списка заметок:

switch (URI\_MATCHER.match(uri)) {

case NOTES:

return null;

default:

return null;

}

В случае, если вставляем элемент в список заметок, "проксируем" запрос в БД:

long rowId = db.insert(NotesContract.Notes.TABLE\_NAME,

null, // nullColumnHack, нам не требуется

contentValues);

Метод возвращает rowId — идентификатор вставленной строки. Если вставка прошла успешно, то значение rowId будет больше нуля.

Теперь нам нужно создать и вернуть URI созданного элемента и оповестить систему, что данные изменились:

if (rowId > 0) {

Uri noteUri = ContentUris.withAppendedId(NotesContract.Notes.URI, rowId);

getContext().getContentResolver().notifyChange(uri, null);

return noteUri;

}

Всё. В итоге код метода будет таким:

@Nullable

@Override

public Uri insert(@NonNull Uri uri, @Nullable ContentValues contentValues) {

SQLiteDatabase db = notesDbHelper.getWritableDatabase();

switch (URI\_MATCHER.match(uri)) {

case NOTES:

long rowId = db.insert(NotesContract.Notes.TABLE\_NAME,

null,

contentValues);

if (rowId > 0) {

Uri noteUri = ContentUris.withAppendedId(NotesContract.Notes.URI, rowId);

getContext().getContentResolver().notifyChange(uri, null);

return noteUri;

}

return null;

default:

return null;

}

}

update() и delete()

Эти методы мы пока не будем трогать, поэтому просто переопределите их:

@Override

public int delete(@NonNull Uri uri, @Nullable String selection, @Nullable String[] selectionArgs) {

return 0;

}

@Override

public int update(@NonNull Uri uri, @Nullable ContentValues contentValues,

@Nullable String selection, @Nullable String[] selectionArgs) {

return 0;

}

Регистрация ContentProvider

ContentProvider, как и любой другой компонент Android-приложения, должен быть зарегистрирован в AndroidManifest.xml.

Добавьте его сразу после закрывающего тега <activity>:

<provider

android:name=".db.NotesProvider"

android:authorities="com.skillberg.notes.provider"

android:exported="false" />

* Атрибут android:name содержит информацию о том, в каком классе находится ContentProvider. Это может быть полный (например, в моём случае — com.skillberg.notes. db.NotesProvider) или относительный (как в примере выше) путь.
* Атрибут android:authorities содержит информацию об **authority** — то, что мы задавали ранее в контракте БД.
* Атрибут android:exported со значением false говорит о том, что у других приложений не будет доступа к нашему ContentProvider.

Проверяем работу

Давайте проверим, что всё работает так, как было задумано — вставим заметку и получим её из базы.

Откройте MainActivity и создайте метод insert():

private void insert() {

}

Теперь нужно написать код, который отвечает за вставку заметки в БД.

Всё взаимодействие с ContentProvider осуществляется при помощи специального объекта ContentResolver. В нём есть методы, аналогичные созданным нами в ContentProvider.

Сначала получим ContentResolver, используя метод getContentResolver():

ContentResolver contentResolver = getContentResolver();

Нам нужен метод ContentResolver.insert(), принимающий два параметра:

* Uri uri — **URI**, по которому вставляем данные.
* ContentValues contentValues — массив пар "ключ:значение" (имя столбца и его значение).

Создадим ContentValues — это делается очень просто:

ContentValues contentValues = new ContentValues();

contentValues.put(NotesContract.Notes.COLUMN\_TITLE, "Заголовок заметки");

contentValues.put(NotesContract.Notes.COLUMN\_NOTE, "Текст заметки");

contentValues.put(NotesContract.Notes.COLUMN\_CREATED\_TS, System.currentTimeMillis());

contentValues.put(NotesContract.Notes.COLUMN\_UPDATED\_TS, System.currentTimeMillis());

И, собственно, производим вставку:

Uri uri = contentResolver.insert(NotesContract.Notes.URI, contentValues);

Как вы помните, метод insert() возвращает **URI** вставленного контента, если вставка произведена успешно. Выведем его в лог:

Log.i("Test", "URI: " + uri);

А теперь вставьте вызов созданного нами метода в конец onCreate(). В итоге получится вот такой код:

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

insert();

}

private void insert() {

ContentResolver contentResolver = getContentResolver();

ContentValues contentValues = new ContentValues();

contentValues.put(NotesContract.Notes.COLUMN\_TITLE, "Заголовок заметки");

contentValues.put(NotesContract.Notes.COLUMN\_NOTE, "Текст заметки");

contentValues.put(NotesContract.Notes.COLUMN\_CREATED\_TS, System.currentTimeMillis());

contentValues.put(NotesContract.Notes.COLUMN\_UPDATED\_TS, System.currentTimeMillis());

Uri uri = contentResolver.insert(NotesContract.Notes.URI, contentValues);

Log.i("Test", "URI: " + uri);

}

}

Запустите приложение и проверьте вывод — в нём должно быть примерно следующее:

10-16 20:57:52.379 2845-2845/? I/Test: URI: content://com.skillberg.notes.provider/notes/6

Работает!

**Обратите внимание**: в данном примере мы получаем доступ к данным из UI-потока.

В "настоящих" приложениях так делать нельзя, так как это может привести к лагам в приложении или ANR.

Мы с вами обязательно исправим эту недоработку в следующих уроках.

Теперь сделаем выборку данных из БД. Создайте метод select():

private void select() {

ContentResolver contentResolver = getContentResolver();

}

Для получения данных используется метод ContentResolver.query(), аналогичный созданному нами ранее в ContentProvider и принимающий такие же аргументы.

Прежде чем мы начнём использовать его, добавим в класс NotesContract.Notes следующее поле:

public static final String[] LIST\_PROJECTION = {

\_ID,

COLUMN\_TITLE,

COLUMN\_CREATED\_TS,

COLUMN\_UPDATED\_TS

};

В этом массиве мы перечисляем столбцы, которые хотим получать из базы данных при выборке массива заметок.

Теперь в методе MainActivity.query() добавьте:

Cursor cursor = contentResolver.query(

NotesContract.Notes.URI, // URI

NotesContract.Notes.LIST\_PROJECTION, // Столбцы

null, // Параметры выборки

null, // Аргументы выборки

null // Сортировка по умолчанию

);

Таким образом мы получили Cursor с данными. Теперь просто выведем количество строк в нашей таблице в логи:

Log.i("Test", "Count: " + cursor.getCount());

И не забудем закрыть курсор:

cursor.close();

Ну и, конечно, добавьте вызов select() в конец onCreate().

В итоге должен получиться такой код MainActivity:

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

insert();

select();

}

private void insert() {

ContentResolver contentResolver = getContentResolver();

ContentValues contentValues = new ContentValues();

contentValues.put(NotesContract.Notes.COLUMN\_TITLE, "Заголовок заметки");

contentValues.put(NotesContract.Notes.COLUMN\_NOTE, "Текст заметки");

contentValues.put(NotesContract.Notes.COLUMN\_CREATED\_TS, System.currentTimeMillis());

contentValues.put(NotesContract.Notes.COLUMN\_UPDATED\_TS, System.currentTimeMillis());

Uri uri = contentResolver.insert(NotesContract.Notes.URI, contentValues);

Log.i("Test", "URI: " + uri);

}

private void select() {

ContentResolver contentResolver = getContentResolver();

Cursor cursor = contentResolver.query(

NotesContract.Notes.URI, // URI

NotesContract.Notes.LIST\_PROJECTION, // Столбцы

null, // Параметры выборки

null, // Аргументы выборки

null // Сортировка по умолчанию

);

Log.i("Test", "Count: " + cursor.getCount());

cursor.close();

}

}

Запустите приложение, и в логах вы увидите примерно такой текст:

10-16 21:16:59.005 4585-4585/? I/Test: URI: content://com.skillberg.notes.provider/notes/7

10-16 21:16:59.006 4585-4585/? I/Test: Count: 7

В следующем уроке мы начнём создавать интерфейс для отображения заметок!

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/NotesAndroid/tree/lesson-21)