Loaders

Loader (загрузчик) — специальный механизм, появившийся в далёком Android 3.0, и позволяющий без труда загружать данные в фоне.

Возможность фоновой загрузки, однако, не единственное преимущество загрузчиков.

Помимо этого они отслеживают источник данных, автоматически подгружая изменения, а так же избавляют нас от уймы лишнего шаблонного кода.

CursorLoader

Давайте перепишем метод MainActivity.select() из прошлого урока, используя загрузчики.

В Android SDK есть класс CursorLoader, созданный специально для работы с БД, в первую очередь — с ContentProvider.

Давайте же разберёмся, как работать с загрузчиками.

В MainActivity в конце метода onCreate() напишите (весь написанный ранее код можно удалить):

getLoaderManager().initLoader(

0, // Идентификатор загрузчика

null, // Аргументы

this // Callback для событий загрузчика

);

* **Идентификатор загрузчика** нужен для того, чтобы LoaderManager мог понять, о каком конкретно загрузчике идёт речь.
* **Аргументы опциональны**, и будут переданы в загрузчик при создании. В нашем случае они не нужны.
* **Callback** нужен для получения событий загрузчика.

На последнем аргументе нужно остановиться поподробнее.

Имеющийся на данный момент код не будет собираться, так как в качестве коллбэка мы передали Activity, а она не реализует интерфейс LoaderManager.LoaderCallbacks.

Давайте реализуем его и необходимые методы. В результате получим такой код:

public class MainActivity extends AppCompatActivity implements LoaderManager.LoaderCallbacks<Cursor> {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

getLoaderManager().initLoader(

0, // Идентификатор загрузчика

null, // Аргументы

this // Callback для событий загрузчика

);

}

@Override

public Loader<Cursor> onCreateLoader(int id, Bundle args) {

return null;

}

@Override

public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {

}

@Override

public void onLoaderReset(Loader<Cursor> loader) {

}

}

Что происходит, когда мы вызываем метод LoaderManager.initLoader()? LoaderManager смотрит, не существует ли уже загрузчик с заданным идентификатором.

Если существует — то использует его. Если же не существует, то создаёт новый Loader, вызывая метод onCreateLoader() из коллбэка, то есть создать загрузчик должны мы сами.

Давайте сделаем это:

@Override

public Loader<Cursor> onCreateLoader(int id, Bundle args) {

return new CursorLoader(

this, // Контекст

NotesContract.Notes.URI, // URI

NotesContract.Notes.LIST\_PROJECTION, // Столбцы

null, // Параметры выборки

null, // Аргументы выборки

null // Сортировка по умолчанию

);

}

Как видите, единственное отличие от ContentResolver в параметрах — контекст.

onLoadFinished

Этот метод вызовется, когда Loader закончит загружать данные. Мы работаем с Cursor, поэтому в качестве параметра будет передан курсор.

Давайте, как и в прошлом уроке, выведем в лог количество записей в таблице:

@Override

public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {

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

}

Как вы могли заметить, закрывать Cursor не нужно — CursorLoader сделает это за нас!

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

10-16 23:03:48.644 5047-5047/? I/Test: Load finished: 7

onLoaderReset

Во многих случаях вам может потребоваться "сбросить" загрузчик, чтобы создать новый, который загрузит другие данные. Соответственно, старые данные станут неактуальными вскоре после сброса загрузчика.

Именно в этот момент и вызывается метод onLoaderReset(). Он говорит о том, что данные уже неактуальны, и нужно перестать их использовать. В нашем случае метод пока что не используется, так что оставим его пустым.

В итоге получим вот такой код:

public class MainActivity extends AppCompatActivity implements LoaderManager.LoaderCallbacks<Cursor> {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

getLoaderManager().initLoader(

0, // Идентификатор загрузчика

null, // Аргументы

this // Callback для событий загрузчика

);

}

@Override

public Loader<Cursor> onCreateLoader(int id, Bundle args) {

return new CursorLoader(

this, // Контекст

NotesContract.Notes.URI, // URI

NotesContract.Notes.LIST\_PROJECTION, // Столбцы

null, // Параметры выборки

null, // Аргументы выборки

null // Сортировка по умолчанию

);

}

@Override

public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {

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

}

@Override

public void onLoaderReset(Loader<Cursor> loader) {

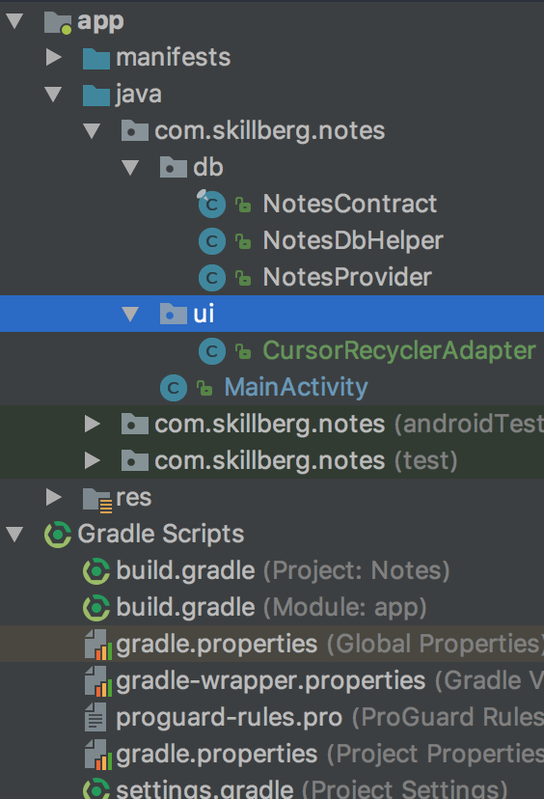
}

}

CursorRecyclerAdapter

Как вы помните, для отображения данных в RecyclerView необходим адаптер. Однако, обычный адаптер нам тут не особо поможет — нужно создать адаптер, который будет работать с Cursor в качестве источника данных.

Давайте реализуем его. Создайте пакет ui, а в нём класс CursorRecyclerAdapter:

[](https://skillberg.com/media/uploads/2017/11/17/cd80447411ff4801ac42e8b05156cc7a.png)CursorRecyclerAdapter в пакете ui

Не забудьте добавить библиотеку с RecyclerView в зависимости build.gradle, находящийся в модулеapp, например, вот так: implementation 'com.android.support:recyclerview-v7:26.1.0'

Сам класс будет аж вот таким (для начала):

public abstract class CursorRecyclerAdapter<ViewHolder extends RecyclerView.ViewHolder>

extends RecyclerView.Adapter<ViewHolder> {

protected Cursor cursor; // Курсор

protected boolean isDataValid; // Валидны ли данные

protected int idColumnIndex; // Индекс столбца ID в курсоре

public CursorRecyclerAdapter(Cursor cursor) {

super();

this.cursor = cursor;

// Данные корректны если курсор не null

isDataValid = cursor != null;

// Пытаемся получить индекс столбца ID, если курсор не null, в ином случае -1

idColumnIndex = cursor != null

? cursor.getColumnIndexOrThrow(NotesContract.Notes.\_ID)

: -1;

// Каждый элемент имеет уникальный ID

setHasStableIds(true);

}

}

Когда мы работаем с курсором, мы двигаемся от строчки к строчке и получаем значения каждого столбца по индексу этого столбца.

Например, если столбец \_id идёт первым, то его индекс будет равен 0, и обращаться к нему мы будем используя индекс 0, а не имя.

Чтобы узнать индекс столбца, нужно вызвать метод getColumnIndex() или getColumnIndexOrThrow(), передав в него имя столбца.

Единственное различие между этими двумя методами в том, что первый в случае отсутствия искомого столбца вернёт -1, а второй бросит исключение IllegalArgumentException.

Поскольку теперь мы взаимодействуем не просто с массивом данных, а с Cursor, создадим новый метод:

public abstract void onBindViewHolder(ViewHolder viewHolder, Cursor cursor);

Он будет вызываться в стандартном onBindViewHolder(), который мы переопределим:

@Override

public void onBindViewHolder(ViewHolder viewHolder, int position) {

// Если данные некорректны — кидаем исключение

if (!isDataValid) {

throw new IllegalStateException("Cursor is not valid!");

}

// Попробовали перейти к определённой строке, но это не получилось

if (!cursor.moveToPosition(position)) {

throw new IllegalStateException("Can not move to position " + position);

}

// Вызываем новый метод

onBindViewHolder(viewHolder, cursor);

}

В переопределённом методе добавляем несколько проверок и вызываем новый метод.

Обратите внимание, что ранее, когда мы работали с адаптерами, в методе onBindViewHolder() мы просто брали элемент из списка по индексу и взаимодействовали с ним. С курсорами это выглядит немного иначе.

Когда нам нужно получить доступ к определённой строке в курсоре, мы должны вызвать метод Cursor.moveToPosition(). В случае, если сделать это не удалось (например, количество строк в курсоре меньше, чем индекс строки, который мы передали), метод вернёт false.

Собственно, после этого мы можем получать значения из курсора.

Следующий шаг — переопределим метод getItemCount(). В нём мы просто вернём количество строк в курсоре (если он есть):

@Override

public int getItemCount() {

if (isDataValid && cursor != null) {

return cursor.getCount();

} else {

return 0;

}

}

Далее — getItemId(). Поскольку у каждого элемента в RecyclerView будет свой уникальный идентификатор, мы вызывали setHasStableIds(true) в конструкторе, и теперь должны переопределить метод, который возвращает идентификатор для элемента по его позиции.

Это достаточно просто:

@Override

public long getItemId(int position) {

// Если с данными всё хорошо и есть курсор

if (isDataValid && cursor != null) {

// Если смогли найти нужную строку в курсоре

if (cursor.moveToPosition(position)) {

// Возвращаем значение столбца ID

return cursor.getLong(idColumnIndex);

}

}

// Во всех остальных случаях возвращаем дефолтное значение

return RecyclerView.NO\_ID;

}

Последний шаг — реализовать метод swapCursor(), который будет заменять старый курсор на новый (например, когда мы сбросили лоадер и загрузили данные заново).

@Nullable

public Cursor swapCursor(Cursor newCursor) {

// Если курсор не изменился — ничего не заменяем

if (newCursor == this.cursor) {

return null;

}

Cursor oldCursor = this.cursor;

this.cursor = newCursor;

if (newCursor != null) {

idColumnIndex = newCursor.getColumnIndexOrThrow(NotesContract.Notes.\_ID);

isDataValid = true;

notifyDataSetChanged();

} else {

idColumnIndex = -1;

isDataValid = false;

// Сообщаем, что данных в адаптере больше нет

notifyItemRangeRemoved(0, getItemCount());

}

return oldCursor;

}

Итоговый код вы можете посмотреть по ссылке в конце урока.

View для заметок

Теперь давайте создадим лэйаут для отображения элементов списка заметок. Назовите его view\_item\_note. Лэйаут будет очень простым: два TextView, в верхнем — заголовок заметки, в нижнем — дата последнего изменения.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView

android:id="@+id/title\_tv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="18sp" />

<TextView

android:id="@+id/date\_tv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="8dp"

android:textSize="14sp" />

</LinearLayout>

Адаптер для заметок

Теперь создадим адаптер, пусть это будет NotesAdapter в пакете ui:

public class NotesAdapter extends CursorRecyclerAdapter<NotesAdapter.ViewHolder> {

public NotesAdapter(Cursor cursor) {

super(cursor);

}

@Override

public void onBindViewHolder(ViewHolder viewHolder, Cursor cursor) {

}

@Override

public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

return null;

}

/\*\*

\* View holder

\*/

class ViewHolder extends RecyclerView.ViewHolder {

public ViewHolder(View itemView) {

super(itemView);

}

}

}

Метод onCreateView() будет выглядеть примерно так же, как и в одном из прошлых уроков:

@Override

public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

LayoutInflater layoutInflater = LayoutInflater.from(parent.getContext());

View view = layoutInflater.inflate(R.layout.view\_item\_note, parent, false);

return new ViewHolder(view);

}

В ViewHolder тоже ничего нового, просто находим нужные View:

class ViewHolder extends RecyclerView.ViewHolder {

private final TextView titleTv;

private final TextView dateTv;

public ViewHolder(View itemView) {

super(itemView);

this.titleTv = itemView.findViewById(R.id.title\_tv);

this.dateTv = itemView.findViewById(R.id.date\_tv);

}

}

А вот с bindViewHolder() будет интереснее. Теперь в нём мы получаем Cursor, который находится на нужной нам строке, и нам нужно получить необходимые данные из этой строки.

Как вы помните, для того, чтобы получить данные из столбца нужно сначала узнать индекс этого столбца. Начнём с заголовка заметки, получим индекс столбца:

int titleColumnIndex = cursor.getColumnIndexOrThrow(NotesContract.Notes.COLUMN\_TITLE);

И теперь получим сам заголовок:

String title = cursor.getString(titleColumnIndex);

Ну и установим его в TextView:

viewHolder.titleTv.setText(title);

Ничего сложного, не так ли?

Точно так же поступаем и с датой, только нужно отформатировать её, используя стандартный SimpleDateFormat, который мы объявим во ViewHolder, дабы не создавать каждый раз:

class ViewHolder extends RecyclerView.ViewHolder {

final SimpleDateFormat SDF = new SimpleDateFormat("dd.MM.yyyy HH:mm", Locale.getDefault());

И в onBindViewHolder() установим дату:

int dateColumnIndex = cursor.getColumnIndexOrThrow(NotesContract.Notes.COLUMN\_UPDATED\_TS);

long updatedTs = cursor.getLong(dateColumnIndex);

Date date = new Date(updatedTs);

viewHolder.dateTv.setText(viewHolder.SDF.format(date));

В итоге получим вот такой метод:

@Override

public void onBindViewHolder(ViewHolder viewHolder, Cursor cursor) {

int titleColumnIndex = cursor.getColumnIndexOrThrow(NotesContract.Notes.COLUMN\_TITLE);

String title = cursor.getString(titleColumnIndex);

viewHolder.titleTv.setText(title);

int dateColumnIndex = cursor.getColumnIndexOrThrow(NotesContract.Notes.COLUMN\_UPDATED\_TS);

long updatedTs = cursor.getLong(dateColumnIndex);

Date date = new Date(updatedTs);

viewHolder.dateTv.setText(viewHolder.SDF.format(date));

}

С адаптером закончили, осталось лишь создать RecyclerView в MainActivity и привязать к нему данные.

RecyclerView для заметок

В activity\_main.xml замените лэйаут на такой:

<?xml version="1.0" encoding="utf-8"?>

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.skillberg.notes.MainActivity">

<android.support.v7.widget.RecyclerView

android:id="@+id/notes\_rv"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</FrameLayout>

В MainActivity инициализируем RecyclerView в onCreate():

private NotesAdapter notesAdapter;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

RecyclerView recyclerView = findViewById(R.id.notes\_rv);

LinearLayoutManager layoutManager = new LinearLayoutManager(this);

recyclerView.setLayoutManager(layoutManager);

DividerItemDecoration dividerItemDecoration = new DividerItemDecoration(this,

layoutManager.getOrientation());

recyclerView.addItemDecoration(dividerItemDecoration);

notesAdapter = new NotesAdapter(null);

recyclerView.setAdapter(notesAdapter);

getLoaderManager().initLoader(

0, // Идентификатор загрузчика

null, // Аргументы

this // Callback для событий загрузчика

);

}

В методе onLoadFinished() заменяем курсор в адаптере:

@Override

public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {

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

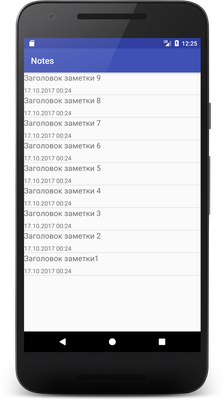
cursor.setNotificationUri(getContentResolver(), NotesContract.Notes.URI);

notesAdapter.swapCursor(cursor);

}

Метод setNotificationUri() нужен для того, чтобы ContentResolver автоматически отслеживал изменения данных и обновлял их (например, когда мы вставляем новую заметку).

Готово! Запустите приложение, и вы увидите, что заметки отображаются:

[](https://skillberg.com/media/uploads/2017/11/17/4f7f5d2e88a5427ca5dda382d43990c0.png)Результат использования CursorLoader с RecyclerView

А в следующем уроке мы будем приводить его к нормальному внешнему виду и просматривать заметки!

ССЫЛКИ

[ГАЙДЛАЙНЫ ПО MATERIAL DESIGN](https://material.io/guidelines/#introduction-principles)

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/NotesAndroid/tree/lesson-22)