Модуль 4. Алгоритмы и структуры данных

Тема 4.11.* Контент-провайдеры в Android

Оглавление

4.11. Контент-провайдеры в Android

Контент-провайдеры предоставляют *интерфейс для публикации и потребления* структурированных наборов данных, основанный на URI[1] с использованием простой схемы *content://*. Их использование позволяет отделить код приложения от данных, делая программу менее чувствительной к изменениям в источниках данных.

Для взаимодействия с контент-провайдером используется уникальный URI, который обычно формируется следующим образом:

```
content://<домен-разработчика-наоборот>.provider.<имя-приложения>/<путь-к-даным>
```

Классы, реализующие контент-провайдеры, чаще всего имеют статическую строковую константу CONTENT_URI, которая используется для обращения к данному контент-провайдеру.

Контент-провайдеры являются единственным способом доступа к данным других приложений и используются для получения результатов запросов, обновления, добавления и удаления данных. Если у приложения есть нужные полномочия, оно может запрашивать и модифицировать соответствующие данные, принадлежащие другому приложению, в том числе данные стандартных БД Android. В общем случае, контент-провайдеры следует создавать только тогда, когда требуется предоставить другим приложениям доступ к данным вашего приложения. В остальных случаях рекомендуется использовать СУБД (SQLite). Тем не менее, иногда контент-провайдеры используются внутри одного приложения для поиска и обработки специфических запросов к данным.

4.11.1. Использование контент-провайдеров

Для доступа к данным какого-либо контент-провайдера используется объект класса *ContentResolver,* который можно получить с помощью метода *getContentResolver* контекста приложения для связи с поставщиком в качестве клиента:

```
ContentResolver cr = getApplicationContext().getContentResolver();
```

Объект *ContentResolver* взаимодействует с объектом контент-провайдера, отправляя ему запросы клиента. Контент-провайдер обрабатывает запросы и возвращает результаты обработки.

Контент-провайдеры представляют свои данные потребителям в виде одной или нескольких таблиц подобно таблицам реляционных БД. Каждая строка при этом является отдельным «объектом» со свойствами, указанными в соответствующих именованных полях. Как правило, каждая строка имеет уникальный целочисленный индекс и именем «**_id**», который служит для однозначной идентификации требуемого объекта.

Контент-провайдеры, обычно предоставляют минимум два URI для работы с данными: один для запросов, требующих все данные сразу, а другой – для обращения к конкретной «строке». В последнем случае в конце URI добавляется /<номер-строки> (который совпадает с индексом «**_id**»).

[1] URI (Uniform Resource Identifier) - унифицированный (единообразный) идентификатор ресурса

Запросы на получение данных

Запросы на получение данных похожи на запросы к БД, при этом используется метод query объекта ContentResolver. Ответ также приходит в виде курсора, «нацеленного» на результирующий набор данных (выбранные строки таблицы):

```
ContentResolver cr = getContentResolver();
    // получить данные всех контактов
    Cursor c = cr.query(ContactsContract.Contacts.CONTENT_URI,          null, null, null, null);

    // получить все строки, где третье поле имеет конкретное
    // значение и отсортировать по пятому полю
    String where = KEY_COL3 + "=" + requiredValue;
    String order = KEY_COL5;

    Cursor someRows = cr.query(MyProvider.CONTENT_URI, null,                    where, null,
order);
```

Изменение данных

Для изменения данных применяются методы insert, update и delete объекта ContentResolver. Для массовой вставки также существует метод bulkInsert.

Пример добавления данных:

```
    ContentResolver cr = getContentResolver();

    ContentValues newRow = new ContentValues();
    // повторяем для каждого поля в строке:
    newRow.put(COLUMN_NAME, newValue);
    Uri myRowUri = cr.insert(SampleProvider.CONTENT_URI, newRow);

    // Массовая вставка:
    ContentValues[] valueArray = new ContentValues[5];

    // здесь заполняем массив
    // делаем вставку
    int count = cr.bulkInsert(MyProvider.CONTENT_URI,                    valueArray);
```

При вставке одного элемента метод insert возвращает URI вставленного элемента, а при массовой вставке возвращается количество вставленных элементов.

Пример удаления:

```
    ContentResolver cr = getContentResolver();
    // удаление конкретной строки
    cr.delete(myRowUri, null, null);

    // удаление нескольких строк
    String where = "_id < 5";
    cr.delete(MyProvider.CONTENT_URI, where, null);
```

Пример изменения:

```
    ContentValues newValues = new ContentValues();

    newValues.put(COLUMN_NAME, newValue);
```

```
String where = "_id < 5";

getContentResolver().update(MyProvider.CONTENT_URI, newValues, where, null);
```

4.11.2. Стандартный контент-провайдер ContactsContract

В Android разработчикам предоставляется возможность получить всю информацию из базы данных контактов для любого приложения с полномочиями READ_CONTACTS.

Начиная с версии 2.0 Android (API level 5) для для хранения и управления контактами на устройстве необходимо использовать провайдер ContactsContract. Это прежде всего связано с тем, что сейчас контакт на мобильном устройстве это не только имя и номер телефона, но и email, Skype, Facebook аккаунт. И при этом пользователю удобно, когда все эти данные привязаны к определенному человеку. Этот контент-провайдер позволяет управлять контактами в Android через базу данных со всей информацией, касающейся контактов.

ContactsContract работает не с одной строго определенной таблицей, отвечающей за хранение данных, а трехуровневой моделью, которая позволяет объединять и связывать данные с конкретным человеком, используя следующие таблицы: contacts, raw_contacts и data.

Data. В исходной таблице каждая строка определяет набор личных данных (например, телефонные номера, адреса электронной почты и т. д.). То, какие именно данные находятся в конкретной строке, определяется с помощью типа MIME, указанного для нее. Универсальные столбцы способны хранить до 15 различных секций с данными разного типа. При добавлении новых данных в таблицу Data, нужно указать объект класса RawContacts для каждого связанного набора данных.

MIME[2] (произн. «майм», англ. Multipurpose Internet Mail Extensions — многоцелевые расширения интернет-почты) — стандарт, описывающий передачу различных типов данных по электронной почте, а также, в общем случае, спецификация для кодирования информации и форматирования сообщений таким образом, чтобы их можно было пересылать по Интернету.

RawContacts. Каждая строка в таблице RawContacts определяет учетную запись, с которой будут ассоциироваться данные из таблицы Data.

Contacts. Эта таблица объединяет все строки из RawContacts, которые относятся к одному и тому же человеку.

На деле используют таблицу Data для добавления, удаления или изменения данных, относящихся к существующим учетным записям, RawContacts — для создания и управления самими учетными записями, Contacts и Data — для получения доступа к базе данных и извлечения информации о контактах.

Важно! По умолчанию, методы Активностей и Фрагментов работают в главном потоке (Main Thread) и должны, в идеале, заканчиваться моментально, т. к. «замораживание» главного потока недопустимо. Для выполнения длительных операций должны использоваться вспомогательные/рабочие потоки (Worker Threads). С другой стороны, результатом запроса к

[2] MIME // Википедия. [2015—2015]. Дата обновления: 04.02.2015. URL: http://ru.wikipedia.org/?oldid=68364168 (дата обращения: 04.02.2015)

Контент-провайдеру является объект типа Cursor, жизненный цикл которого обычно не совпадает с жизненным циклом Активности или Фрагмента, где этот курсор используется. Для «автоматизации» решения обеих задач (выполнения длительных действий и управления курсорами) в Android широко используются Загрузчики (Loaders).

4.11.3. Загрузчики (Loader)

Представленные впервые в API 11, Загрузчики упрощают асинхронную загрузку данных в Активностях и Фрагментах. Они обладают следующими характеристиками:

- Доступны в любой Активности и Фрагменте;
- Обеспечивают асинхронную загрузку данных;
- Отслеживают источники данных и доставляют новые данные при их изменении;
- Автоматически переустанавливают соединение с последним используемым Курсором, когда пересоздаются при изменении конфигурации, таким образом устраняя необходимость выполнять повторные запросы.

При использовании Загрузчиков применяются несколько классов и интерфейсов:

LoaderManager: абстрактный класс, связанный с Активностью или Фрагментом, используется для управления объектами класса Loader. Он помогает приложению управлять долговременными операциями в привязке к жизненному циклу Активности или Фрагмента. Наиболее часто LoaderManager применяется с CursorLoader, но может быть реализован для загрузки любых данных. Для каждой Активности или фрагмента может существовать только один LoaderManager, который может использовать несколько объектов Loader.

LoaderManager.LoaderCallbacks: методы, реализующие клиентский интерфейс для взаимодействия с LoaderManager. Например, метод onCreateLoader() применяется для создания нового Загрузчика.

Loader: абстрактный класс, осуществляющий асинхронную загрузку данных, он является базовым классом для Загрузчика. Чаще всего используется подкласс CursorLoader.

AsyncTaskLoader: абстрактный класс, обеспечивающий AsyncTask для выполнения нужных действий.

CursorLoader: подкласс AsyncTaskLoader, который делает запрос к ContentResolver и возвращает Курсор. Этот класс создает AsyncTaskLoader для выполнения запросов в фоновом потоке, не блокируя UI приложения. Использование этого Загрузчика является наилучшим способом асинхронной загрузки данных из Контент-Провайдеров.

В настоящее время Загрузчики доступны и в API < 11, если используются Библиотеки совместимости (Библиотеки поддержки, Support Libraries).

В приложении, применяющем Загрузчики, обычно используются следующие объекты:

- Активность или Фрагмент;
- Объект класса LoaderManager;
- CursorLoader, связанный с Контент-Провайдером, либо собственная реализация подклассов Loader или AsyncTask для загрузки данных из других источников;
- Реализация LoaderManager.CallBacks, в которой будут создаваться новые Загрузчики и отслеживаться ссылки на уже существующие;
- Реализация отображения данных, например, с помощью SimpleCursorAdapter;
- Источник данных, такой, как ContentProvider, если используется CursorLoader.

Запуск и перезапуск Загрузчика

Обычно Loader инициализируется в методе onCreate() Активности или методе onActivityCreated() Фрагмента. Делается это примерно так:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      mAdapter = new SimpleCursorAdapter(this,
      R.layout.item, null,       FROM, TO, 0);

      setListAdapter(mAdapter);
      getLoaderManager().initLoader(0, null, this);
}
```

Метод initLoader() обеспечивает создание Загрузчика и/или его активизацию: если Загрузчик с таким идентификатором уже имеется, он будет повторно использован, в противном случае будет создан новый Загрузчик (путем вызова метода onCreateLoader() интерфейса LoaderManager.LoaderCallbacks).

У метода initLoader() на примере выше были использованы следующие аргументы:

- идентификатор Загрузчика (0)
- дополнительные аргументы, передаваемые Загрузчику при создании (null)
- ссылка на объект, реализующий интерфейс LoaderManager.LoaderCallbacks (this)

Если требуется обязательно создать новый Загрузчик (например, потому, что данные уже имеющегося Загрузчика могут быть устаревшими), вместо initLoader() используется метод restartLoader().

Интерфейс LoaderManager.LoaderCallbacks

Данный интерфейс позволяет клиентам взаимодействовать с объектом *LoaderManager* и содержит три метода: *onCreateLoader()* (создает и возвращает новый Загрузчик с заданным идентификатором), *onLoadFinished()* (вызывается, когда Загрузчик завершает загрузку) и *onLoadReset()* (вызывается, когда Загрузчик сбрасывается и его данные больше не используются и их можно освободить).

Ниже показаны примеры реализации этих методов:

```
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle arg){

    return new CursorLoader(this, ContactsContract
          .CommonDataKinds.Phone.CONTENT_URI,
    PROJECTION, null, null, null);
    }

    @Override
    public void onLoadFinished(Loader<Cursor> loader,
    Cursor data) {

          mAdapter.swapCursor(data);
    }

    @Override
```

```
    public void onLoaderReset(Loader<Cursor> loader) {
          mAdapter.swapCursor(null);
    }
}
```

Как видно из примеров, методы эти весьма просты, но стоит обратить внимание на метод *swapCursor()* используемого адаптера: с помощью этого метода предотвращается дальнейшее использование старых данных, доступных через курсор (если они имелись), но сам курсор остается открытым – он будет закрыт Загрузчиком класса *CursorLoader*, который его создал (в конструкторе, вызванном в методе *onCreateLoader()*) и отслеживает его жизненный цикл, в том числе, уничтожая при необходимости.

Дополнительную информацию (включая примеры), в том числе о других типах Загрузчиков, можно найти в описаниях классов:

- Loader: http://developer.android.com/reference/android/content/Loader.html
- AsyncTaskLoader: http://developer.android.com/reference/android/content/AsyncTaskLoader.html
- LoaderManager: http://developer.android.com/reference/android/app/LoaderManager.html

Упражнение 4.11.1.

Суть данного упражнения в том, чтобы отобразить в приложении список имен контактов из адресной книги Андроид.

Создадим новый Андроид проект, назовем его ContactsSample. В первую очередь, отредактируем AndroidManifest.xml, так как для чтения контактов нужно соответствующее разрешение. Добавим строку

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

в этот файл, в результате, его содержимое должно выглядеть примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ru.samsung.itschool.contactssample" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category  android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>
```

```
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
</manifest>
```

В проекте будет одна основная активность - назовем ее MainActivity. Имеет смысл унаследовать ее от стандартного класса FragmentActivity, чтобы можно было использовать Загрузчики. Activity будет содержать один ListView, который и будет использован для отображения списка контактов.

Класс активности должен принять примерно следующий вид:

```
public class MainActivity extends FragmentActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ListView listView = (ListView) findViewById(R.id.contactsList);
        // TODO: отобразить в listView контакты
    }
}
```

Xml-файл разметки для активности должен выглядеть примерно следующим образом:

```
<RelativeLayout
  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"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin"
  tools:context=".MainActivity">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/contactsList">
    </ListView>

</RelativeLayout>
```

Можно было бы не создавать файл разметки, если нам нужно создать только один ListView, однако, если приложение планируется развивать и в активности будут другие View, удобно задавать параметры и взаимное расположение этих view в xml-файле разметки, поэтому, мы воспользуемся именно этим способом.

Чтобы отобразить что-то в ListView, необходим Адаптер. В данном случае, мы можем использовать SimpleCursorAdapter, который позволяет получать данные из Курсора (контент провайдеры предоставляют доступ к данным именно в виде Курсора - объекта, из которого данные можно прочесть). Объявим поле для адаптера, создадим его в методе onCreate активности и назначим его listView:

```
    private SimpleCursorAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ListView listView = (ListView) findViewById(R.id.contactsList);

        adapter = new SimpleCursorAdapter(this,
                    android.R.layout.simple_list_item_2, null, new String[] {
                                Contacts.DISPLAY_NAME,
Contacts.TIMES_CONTACTED },
                    new int[] { android.R.id.text1, android.R.id.text2 }, 0);

        listView.setAdapter(adapter);
        // TODO: загрузить список контактов в адаптер
    }
```

Остановимся более подробно на параметрах конструктора SimpleCursorAdapter. Первым параметром передается текущая Активность, так как адаптеру нужен Контекст для создания отдельных view для элементов списка. Следующий параметр - идентификатор ресурса, соответствующего файлу разметки для одного элемента, который будет отображаться в ListView. Можно было бы создать свой ресурс со своей разметкой, но в этом упражнении мы воспользуемся встроенным андроидовским файлом разметки с идентификатором android.R.layout.simple_list_item_2. Этот файл разметки содержит два текстовых поля. Выглядит он примерно следующим образом (так как он есть в Андроиде, нет необходимости копировать его содержимое в проект):

```
<TwoLineListItem xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="?android:attr/listPreferredItemHeight"
    android:mode="twoLine"
    android:paddingStart="?android:attr/listPreferredItemPaddingStart"
    android:paddingEnd="?android:attr/listPreferredItemPaddingEnd">

      <TextView android:id="@android:id/text1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
            android:layout_marginTop="8dip"
      android:textAppearance="?android:attr/textAppearanceListItem" />

      <TextView android:id="@android:id/text2"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_below="@android:id/text1"
            android:layout_alignStart="@android:id/text1"
      android:textAppearance="?android:attr/textAppearanceListItemSecondary"
/>

</TwoLineListItem>
```

Как можно заметить, по сути, он состоит из двух TextView с идентификаторами text1 и text2 с отличающимися настройками отображения (текст в text1 крупнее). TwoLineListItem является наследником обычного RelativeLayout. Вернемся, однако, к конструктору SimpleCursorAdapter. Очередной - третий - параметр это Курсор, из которого адаптер будет читать данные. Так как в момент вызова метода onCreate курсора у нас еще нет - он будет получен с помощью Загрузчика в

отдельном потоке и его получение может занять ненулевое время, в качестве параметра передан null.

Следующие два параметра - это массив "from" и массив "to". Первый из них - массив строк - содержит список названий колонок Курсора, которые должны быть отображены. Второй массив - это массив идентификаторов тех View, в которых должны быть отображены значения из колонок первого массива. Таким образом, передав в качестве первого массива `new String[] { Contacts.DISPLAY_NAME, Contacts.TIMES_CONTACTED }`, а в качестве второго `new int[] { android.R.id.text1, android.R.id.text2 }`, можно добиться того, чтобы во view с идентификатором text1 отображалось имя контакта, а во View с идентификатором text2 - количество раз, которое пользователь связывался с этим контактом.

Последний параметр конструктора это набор дополнительных флагов, значение которых можно узнать в документации. В этом упражнении они нам не нужны, передадим 0 в качестве этого параметра.

После того, как адаптер создан и назначен ListView, нужно каким-то образом получить данные о контактах в виде курсора, из которого можно читать данные о контактах, и передать этот курсор в адаптер. Для этого используем Загрузчик. Менеджеру загрузчиков нужно будет передать `LoaderManager.LoaderCallbacks`, чтобы он оповещал нас о процессе загрузки, поэтому, реализуем этот интерфейс в нашем классе активности. В качестве шаблонного типа используем Cursor, так как именно его мы собираемся загружать с помощью Загрузчика. Реализация методов интерфейса `LoaderManager.LoaderCallbacks` довольно простая:

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

		// ...

		@Override
		public Loader<Cursor> onCreateLoader(int id, Bundle args) {
			return new CursorLoader(this, Contacts.CONTENT_URI, new
String[] {
					Contacts._ID, Contacts.DISPLAY_NAME,
					Contacts.TIMES_CONTACTED, Contacts.PHOTO_ID },
null, null,
					null);
		}

		@Override
		public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
			adapter.swapCursor(data);
		}

		@Override
		public void onLoaderReset(Loader<Cursor> loader) {
			adapter.swapCursor(null);
		}
	}
```

Метод onCreateLoader будет вызван при необходимости создать новый курсор. Параметр id может быть проигнорирован, так как в этом упражнении используется только один курсор; если бы их было несколько, в зависимости от значения этого параметра нужно было бы создавать нужный курсор. В реализации метода мы просто создаем новый Загрузчик - CursorLoader, передавая ему контекст, uri для отправки запроса контент провайдеру и "проекцию" - массив с

именами полей, которые мы хотим получить в результатах. Стоит отметить, что этот массив полей должен содержать все поля, которые будут отображаться в ListView (иначе, у ListView не будет каких-то данных для отображения), но он не обязан совпадать с ним полностью; например, в коде выше дополнительно запрашивается поле Contacts.PHOTO_ID, которое не отображается в ListView. Кроме того, в этот набор обязательно должен входить идентификатор записи (в данном случае Contacts._ID, который соответствует строке `"_id"`). Оставшиеся параметры отвечают за фильтрацию списка результатов (например, если мы хотим показать только определенные контакты, а не все) и за их порядок; в этом упражнении они нас не интересуют, поэтому, мы передаем null в качестве этих параметров.

Метод onLoadFinished вызывается, когда загрузчик закончил свою работу и имеется готовый курсор. Этот курсор просто передается в соответствующий метод адаптера, который уже позаботится о том, чтобы отобразить необходимые данные в ListView. Если до этого в адаптере был предыдущий курсор, о его освобождении позаботится загрузчик, поэтому, мы используем метод swapCursor для замены предыдущего курсора (если он был) на новый и нигде не вызываем метод close у курсора - об этом заботится загрузчик.

Метод onLoaderReset вызывается, когда загрузчик "сбрасывается", например потому, что менеджер загрузчиков хочет повторно использовать этот загрузчик для какой-то другой задачи. Нужно забыть все ссылки на данные этого загрузчика, так как скоро они станут недоступны, что и реализовано путем передачи null в адаптер в качестве курсора.

Наконец, необходимо инициировать создание загрузчика в методе onCreate, добавив в конец этого метода строку

```
getSupportLoaderManager().initLoader(0, null, this);
```

В качестве идентификатора загрузчика мы передаем 0 (реализация метода onCreateLoader игнорирует идентификатор в любом случае), а в качестве последнего параметра передаем this, то есть, ссылку на активность, которая и реализует методы onCreateLoader, onLoadFinished и onLoaderReset, чтобы менеджер загрузчиков смог их вызвать в нужный момент.

В результате, если вы запустите получившееся приложение на Андроид устройстве, вы сможете увидеть список контактов и число - сколько раз с каждым из этих контактов связывались. Если на вашем тестовом устройстве книга контактов пустая, создайте несколько контактов перед запуском, чтобы проверить работу приложения; достаточно заполнять только имя контакта, так как именно оно выводится в этом упражнении.

Дополнительно, можно выделить в виде именованных констант громоздкие массивы, которые используются в качестве параметров для конструктора класса SimpleCursorAdapter и конструктора класса CursorLoader.

4.11.4. Стандартный контент-провайдер MediaStore

В Android разработчик имеет возможность работать со всем медиа-контенте на мобильном устройстве. Для этого используется класс MediaStore, который является мощным провайдером данных и содержит централизованную базу данных мультимедиа (аудиофайлы, видеофайлы и изображения), размещённых во внутренней памяти устройства или на внешнем носителе (SD-карте).

Упражнение 4.11.2.

Суть данного упражнения в том, чтобы отобразить список миниатюр изображений, размещенных на устройстве. Для этого, необходимо будет воспользоваться провайдером MediaStore.Images.Thumbnails.

Создадим новый Андроид проект с названием **MediaStoreProviderSample**. Снова нужно отредактировать AndroidManifest.xml, так как для работы с файлами на диске нужно соответствующее разрешение. Добавим строку

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

в этот файл, в результате, его содержимое должно выглядеть примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ru.samsung.itschool.mediastoreproviersample" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category  android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
</manifest>
```

Структура проекта будет по большей части совпадать со структурой проекта из упражнения 4.11.1. В проекте снова будет одна основная активность - MainActivity, унаследованная от стандартного класса FragmentActivity для использования Загрузчиков. Activity будет содержать один ListView, который и будет использован для отображения списка.

Приведем примерное содержимое класса Активности (MainActivity.java) и соответствующего ей файла разметки (activity_main.xml):

```
public class MainActivity extends FragmentActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ListView listView = (ListView) findViewById(R.id.imageList);

        // TODO: реализовать отображение списка миниатюр
    }
}
```

```
<RelativeLayout
  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"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin"
  tools:context=".MainActivity">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/imageList">
    </ListView>

</RelativeLayout>
```

В этом упражнении мы создадим свой файл разметки для элемента списка, который будет содержать ImageView для показа миниатюры изображение и TextView с информацией о пути к данной миниатюре. Для этого создадим файл с именем my_item.xml в каталоге layout (рядом с файлом activity_main.xml), в котором и опишем уже упомянутые ImageView и TextView. Вы можете расположить эти View иначе или изменить их размеры. Так или иначе, содержимое файла my_item.xml должно стать примерно следующим:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="96dp">

    <ImageView
        android:layout_margin="2dp"
        android:layout_width="96dp"
        android:layout_height="match_parent"
        android:id="@+id/imageView"
        android:contentDescription="image preview" />

    <TextView
        android:paddingLeft="8dp"
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
```

```
            android:id="@+id/textView"
        />
    </LinearLayout>
```

Теперь можно перейти к созданию адаптера в методе onCreate Активности. Создание Адаптера аналогично упражнению 4.11.1 за тем исключением, что мы используем свой ресурс для разметки элемента (my_item) и свои view, указанные в этом ресурсе (imageView и textView) для отображения данных.

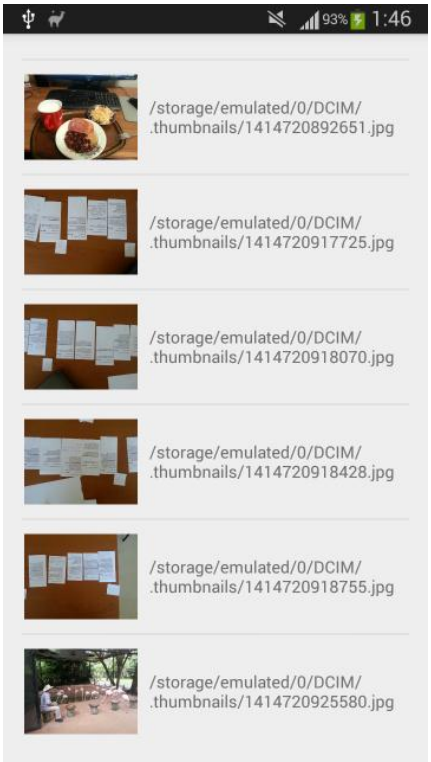
```
public class MainActivity extends FragmentActivity {

    private SimpleCursorAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ListView listView = (ListView) findViewById(R.id.imageList);

        adapter = new SimpleCursorAdapter(this, R.layout.my_item, null,
                    new String[] {
                        MediaStore.Images.Thumbnails.DATA,
                        MediaStore.Images.Thumbnails.DATA
                    },
                    new int[] {
                        R.id.imageView,
                        R.id.textView
                    }, 0);
        listView.setAdapter(adapter);
        // TODO: добавить загрузчик, который получит данные о
        // миниатюрах изображений на устройстве, и передать
        // эти данные в адаптер
    }
```

Мы будем отображать одно и то же содержимое в обоих view - непосредственно uri данной миниатюры (колонку с именем MediaStore.Images.Thumbnails.DATA). Класс SimpleCursorAdapter правильно покажет это uri в виде изображения при отображении на ImageView и в виде текста при отображении на TextView (SimpleCursorAdapter не поддерживает других типов View кроме TextView и ImageView, но можно написать свою реализацию адаптера, если необходимо отображать данные более сложной структуры).

Далее, остается лишь закончить проект, добавив Загрузчик, который будет получать необходимые данные и передавать их адаптеру. Этот процесс подробно описан в упражнении 4.11.1. Стоит отметить, что в качестве uri для запроса мы используем MediaStore.Images.Thumbnails.EXTERNAL_CONTENT_URI, а в качестве “проекции” (набора читаемых колонок) колонки с идентификаторами MediaStore.Images.Thumbnails._ID и MediaStore.Images.Thumbnails.DATA. Приведем окончательный вид класса MainActivity:

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

		private SimpleCursorAdapter adapter;

		@Override
		protected void onCreate(Bundle savedInstanceState) {
			super.onCreate(savedInstanceState);
			setContentView(R.layout.activity_main);

			ListView listView = (ListView) findViewById(R.id.imageList);

			adapter = new SimpleCursorAdapter(this, R.layout.my_item, null,
					new String[] { MediaStore.Images.Thumbnails.DATA,
							MediaStore.Images.Thumbnails.DATA },
new int[] {
							R.id.imageView, R.id.textView }, 0);

			listView.setAdapter(adapter);
			getSupportLoaderManager().initLoader(0, null, this);
		}

		@Override
		public Loader<Cursor> onCreateLoader(int id, Bundle args) {
			return new CursorLoader(this,
					MediaStore.Images.Thumbnails.EXTERNAL_CONTENT_URI,
					new String[] { MediaStore.Images.Thumbnails._ID,
							MediaStore.Images.Thumbnails.DATA },
null, null,
					null);
		}

		@Override
		public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
			adapter.swapCursor(data);
		}

		@Override
		public void onLoaderReset(Loader<Cursor> loader) {
			adapter.swapCursor(null);
		}
	}
```

В результате, если у вас имеется несколько фотографий на устройстве, после запуска приложения вы сможете увидеть их миниатюры в виде списка:



Задание 4.11.1

Вывести имена всех контактов, упорядоченные по времени последнего сделанного ему звонка.

* Вывести дату последних 10 картинок, сохраненных на телефоне.

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю курсов по Android Центра “Специалист” при МГТУ им. Баумана Варакину Михаилу Владимировичу, преподавателю IT ШКОЛЫ SAMSUNG Козловскому Павлу Александровичу.