Маленькая хитрость для отображения большого объёма данных в ListView

В этой статье я хочу поделиться недавно найденным решением, позволяющем отображать, а главное легко скролить большие объёмы данных в стандартном ListView.

Проблема

Стандартный механизм отображения списков из базы данных в Android выглядит примерно так:

* Activity содержит ListView
* ListView обращается к экземпляру CursorAdapter
* CursorAdapter получает данные из объекта, реализующего интерфейс Cursor
* Cursor получен либо из ContentProvider, либо сразу из SQLiteDatabase

Всё работает нормально ровно да тех пор, пока количество строк в Cursor сравнительно небольшое. Но если в нём 50 тысяч, 100 тысяч и более строк (хотя дело не только в количестве строк, но об этом чуть позже), время от времени список будет притормаживать. Особенно это заметно при «быстрой прокрутке», если у ListView установлено в true свойство [fastScrollEnabled](http://developer.android.com/reference/android/widget/AbsListView.html" \l "attr_android:fastScrollEnabled).  
  
Оставим за скобками, почему же нам всё-таки нужно, чтобы в ListView помещалось такое огромное количество данных. Будем считать это требованием заказчика, на которое мы повлиять не в состоянии. Так же будем считать невозможными воркэраунды с прелоадерами в духе Твиттера и «бесконечных списков», следующая порция данных в которых подгружается при достижении конца уже загруженных данных.  
  
Нам нужно, чтобы в любой момент можно было подскролить к любому из ста тысяч элементов списка без заметных подвисаний интерфейса. Как же это сделать? Давайте попробуем, для начала, найти причину тормозов.

Причина

Рассматривать [ViewHolder](http://xjaphx.wordpress.com/2011/06/16/viewholder-pattern-caching-view-efficiently/) мы не будем — я предполагаю, что любой мало-мальски грамотный android-разработчик знает и использует этот паттерн. О недопустимости создания большого количества объектов в методе [getView](http://developer.android.com/reference/android/widget/Adapter.html" \l "getView%28int,%20android.view.View,%20android.view.ViewGroup%29) в силу неизбежности возмездия в лице сборщика мусора я тоже промолчу.  
  
Нас интересует работа курсора к базе данных.  
  
Cursor, который мы получаем из [SQLiteDatabase](http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html), является экземпляром класса [SQLiteCursor](http://developer.android.com/reference/android/database/sqlite/SQLiteCursor.html), который наследуется от [AbstractWindowedCursor](http://developer.android.com/reference/android/database/AbstractWindowedCursor.html). Этот класс же, в свою очередь, содержит в себе экземпляр[CursorWindow](http://developer.android.com/reference/android/database/CursorWindow.html).  
  
В последнем классе как раз и кроется наша проблема. Если вы взгляните на [исходники CursorWindow](http://grepcode.com/file/repository.grepcode.com/java/ext/com.google.android/android/4.1.1_r1/android/database/CursorWindow.java/), то увидите, что размер окна ограничен константой с именем com.android.internal.R.integer.config\_cursorWindowSize. Пользовательский интерфейс притормаживает ровно в тот момент, когда место в окне заканчивается (имеет значение не только количетсво строк в выборке, но и длина каждой строки), и AbstractWindowedCursor запрашивает данные для нового окна, а затем их в это окно копирует.  
  
Можно, конечно, попытаться увеличить размер окна. Но это, во-первых, плохое решение, так как не устранит проблему, а только отдалит её. Во-вторых мы не можем увеличивать его постоянно, так как память устройства ограничена. Ну а в третьих, технически это неоправданно сложно.  
  
Мы пойдём другим путём.

Решение

Вообще говоря, SQLite — достаточно быстрая база данных, и большая часть «тормозов» вызвана неправильным её использованием. Особенно быстро она работает при запросах по первичному ключу.  
  
Идея состоит в следующем: мы запрашиваем только первичные ключи, а затем, при отображении каждой из строк, запрашиваем остальные столбцы по этому первичному ключу. И это на самом деле работает быстрее.  
  
Для иллюстрации этой идеи я написал свою реализацию класса [BaseAdapter](http://developer.android.com/reference/android/widget/BaseAdapter.html).

**package** me.ilich.fastscroll;

**import** android.content.Context;

**import** android.database.Cursor;

**import** android.view.View;

**import** android.view.ViewGroup;

**import** android.widget.BaseAdapter;

**public** **abstract** **class** **QuickAdapter** **extends** **BaseAdapter** {

**private** **final** DataSource mDataSource;

**private** **int** mSize = 0;

**private** Cursor mRowIds = **null**;

**private** **final** Context mContext;

**public** **QuickAdapter**(Context context, DataSource dataSource){

mDataSource = dataSource;

mContext = context;

doQuery();

}

**private** **void** **doQuery**(){

**if**(mRowIds!=**null**){

mRowIds.close();

}

mRowIds = mDataSource.getRowIds();

mSize = mRowIds.getCount();

}

**@Override**

**public** **int** **getCount**() {

**return** mSize;

}

**@Override**

**public** Object **getItem**(**int** position) {

**if**(mRowIds.moveToPosition(position)){

**long** rowId = mRowIds.getLong(0);

Cursor c = mDataSource.getRowById(rowId);

**return** c;

}**else**{

**return** **null**;

}

}

**@Override**

**public** **long** **getItemId**(**int** position) {

**if**(mRowIds.moveToPosition(position)){

**long** rowId = mRowIds.getLong(0);

**return** rowId;

}**else**{

**return** 0;

}

}

**@Override**

**public** View **getView**(**int** position, View convertView, ViewGroup parent) {

mRowIds.moveToPosition(position);

**long** rowId = mRowIds.getLong(0);

Cursor cursor = mDataSource.getRowById(rowId);

cursor.moveToFirst();

View v;

**if** (convertView == **null**) {

v = newView(mContext, cursor, parent);

} **else** {

v = convertView;

}

bindView(v, mContext, cursor);

cursor.close();

**return** v;

}

**public** **abstract** View **newView**(Context context, Cursor cursor, ViewGroup parent);

**public** **abstract** **void** **bindView**(View view, Context context, Cursor cursor);

**public** **interface** **DataSource** {

Cursor **getRowIds**();

Cursor **getRowById**(**long** rowId);

}

}

Для использования этого класса нужно реализовать методы newView и bindView точно так же, как это делается для CursorAdapter, а так же написать реализацию QuickAdapter.DataSource, например так:

**class** **MyDataSource** **implements** **QuickAdapter**.**DataSource** {

**@Override**

**public** Cursor **getRowIds**() {

**return** mDatabase.rawQuery("SELECT rowid FROM table1", **new** String[]{});

}

**@Override**

**public** Cursor **getRowById**(**long** rowId) {

**return** mDatabase.rawQuery("SELECT \* FROM table1 WHERE rowid = ?", **new** String[]{Long.toString(rowId)});

}

}

Заключение

На Samsung Galaxy Tab 10.1 без каких-либо заметных тормозов работал «быстрый скрол» для списка из 300 тысяч элементов, каждый из которых до 2Кб. Стандартный CursorAdapter же тормозил так, что смотреть было страшно.  
  
Для CursorAdapter максимальная задержка вызова метода getView составила 553 мс, для QuickAdapter — 47 мс. Замеры производительности выполнялись с помощью следующего кода:

**@Override**

**public** View **getView**(**int** arg0, View arg1, ViewGroup arg2) {

**long** t1 = System.currentTimeMillis();

View result = **super**.getView(arg0, arg1, arg2);

**long** t2 = System.currentTimeMillis();

**long** dt = t2-t1;

**if**(dt>10){

Log.i("QuickAdapter", dt+"");

}

**return** result;

}

Идея, изложенная здесь, взята из статьи [Нестандартный подход к «повышению производительности» select-запросов в SQLite](http://habrahabr.ru/post/154817/) [Сергея Славина](http://habrahabr.ru/users/shuslav/).W