CODEPATH (/android)

Search

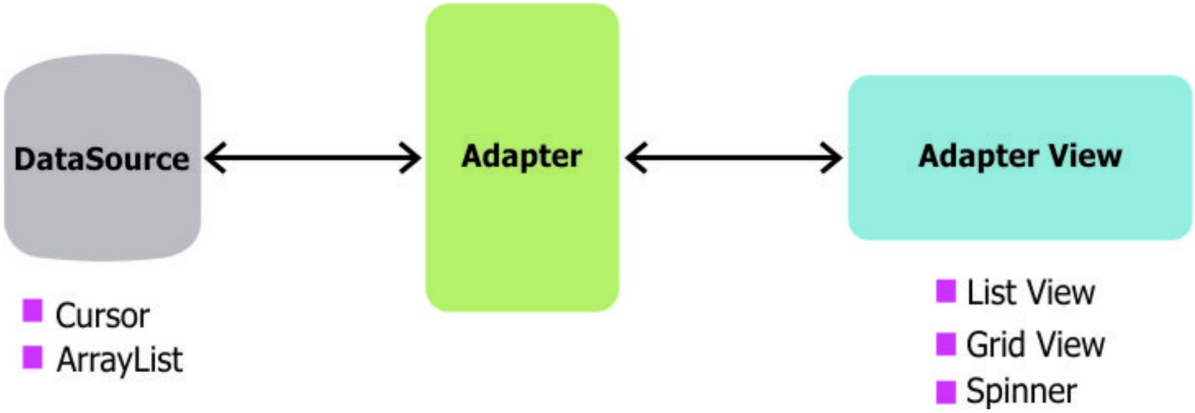
Populating a ListView with a CursorAdapter

Edit Page (https://github.com/codepath/android_guides/wiki/Populating-a-ListView-with-a-CursorAdapter/_edit)

Page History (https://github.com/codepath/android_guides/wiki/Populating-a-ListView-with-a-CursorAdapter/_history)

Overview

In Android development, any time you want to show a vertical list of items you will want to use a ListView which is populated using an Adapter to a data source. When we want the data for the list to be sourced directly from a SQLite database (/android/Persisting-Data-to-the-Device#sqlite) query, we can use a CursorAdapter (http://developer.android.com/reference/android/widget/CursorAdapter.html).



The `CursorAdapter` fits in between a `Cursor` (data source from SQLite query) and the `ListView` (visual representation) and configures two aspects:

- Which layout template to inflate for an item
- Which fields of the cursor to bind to which views in the template

Using a Custom CursorAdapter

Defining our Table

First, we need to define a table within the database from which we will load our cursor. In this case, we will define a database table called **todo_items** for a collection of todo items with a string **body** and an integer **priority**.

body	priority
Get milk	2
Do laundry	3

To create this database table, we would use SQLite persistence (/android/Persisting-Data-to-the-Device#sqlite) or an ORM that allows us to define objects mapped to tables.

Creating the View Template

When we want to display a series of items into a list, using a custom representation of the items, we need

to use our own custom XML layout template for each item. We can simply create an XML layout template in `res/layout/item_todo.xml`, representing a particular cursor row:

```
<?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="horizontal" >
    <TextView
        android:id="@+id/tvBody"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Study cursors"
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <TextView
        android:id="@+id/tvPriority"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:text="3"
        android:textAppearance="?android:attr/textAppearanceMedium" />
</LinearLayout>
```

Defining the Adapter

Next, we need to define the adapter to describe the process of projecting the `Cursor`'s data into a View. To do this we need to override the `newView` method and the `bindView` method. The naive approach to this (without any view caching) looks like the following:

```
public class TodoCursorAdapter extends CursorAdapter {
  public TodoCursorAdapter(Context context, Cursor cursor) {
      super(context, cursor, 0);
  }

  // The newView method is used to inflate a new view and return it,
  // you don't bind any data to the view at this point.
  @Override
  public View newView(Context context, Cursor cursor, ViewGroup parent) {
      return LayoutInflater.from(context).inflate(R.layout.item_todo, parent, false);
  }

  // The bindView method is used to bind all data to a given view
  // such as setting the text on a TextView.
  @Override
  public void bindView(View view, Context context, Cursor cursor) {
      // Find fields to populate in inflated template
      TextView tvBody = (TextView) view.findViewById(R.id.tvBody);
      TextView tvPriority = (TextView) view.findViewById(R.id.tvPriority);
      // Extract properties from cursor
      String body = cursor.getString(cursor.getColumnIndexOrThrow("body"));
      int priority = cursor.getInt(cursor.getColumnIndexOrThrow("priority"));
      // Populate fields with extracted properties
      tvBody.setText(body);
      tvPriority.setText(String.valueOf(priority));
  }
}
```

First, we define a constructor that passes the cursor and context to the superclass. Next, we override the `newView` method, which is used to inflate a new view template. Finally, we override the `bindView` method,

Jump to Section

which is used to bind all data to a given view to populate the template content for the item.

Retrieving the Cursor

In order to use a `CursorAdapter`, we need to query a SQLite database and get back a `Cursor` representing the result set. This requires us to use a `SQLiteOpenHelper` for persistence as described here (/android/Persisting-Data-to-the-Device#sqlite) or an ORM that provides access to the underlying database.

Once you have a database and tables defined, then we can get access to a `Cursor` by querying the database with `rawQuery`:

```
// TodoDatabaseHandler is a SQLiteOpenHelper class connecting to SQLite
TodoDatabaseHandler handler = new TodoDatabaseHandler(this);
// Get access to the underlying writeable database
SQLiteDatabase db = handler.getWritableDatabase();
// Query for items from the database and get a cursor back
Cursor todoCursor = db.rawQuery("SELECT  * FROM todo_items", null);
```

Attaching the Adapter to a ListView

Now, we can use the `CursorAdapter` in the `Activity` to display an array of items into the `ListView`:

```
// Find ListView to populate
ListView lvItems = (ListView) findViewById(R.id.lvItems);
// Setup cursor adapter using cursor from last step
TodoCursorAdapter todoAdapter = new TodoCursorAdapter(this, todoCursor);
// Attach cursor adapter to the ListView
lvItems.setAdapter(todoAdapter);
```

This will then trigger the `CursorAdapter` iterating through the result set and populating the list. We can change the cursor to update the adapter at any time with:

```
// Switch to new cursor and update contents of ListView
todoAdapter.changeCursor(todoCursor);
```

References

- http://www.gustekdev.com/2013/05/custom-cursoradapter-and-why-not-use.html (http://www.gustekdev.com/2013/05/custom-cursoradapter-and-why-not-use.html)

(https://github.com/codepath/android_guides/wiki/)