ListActivity - создаём прокручиваемый список

Список за пять минут

Очень часто экран приложения состоит из обычного прокручиваемого списка. Например, это может быть список контактов, дни месяца, ассортимент товара, технические характеристики модели и так далее. Android позволяет создать такой список за пару минут.

В предыдущих примерах мы встречали в коде строчку **public class HelloWorld extends Activity**, что означало наследование от специального класса **Activity** или производных классов, например, **AppCompatActivity**. Существует ещё один специальный класс **ListActivity**, специально разработанный для списков.

Сейчас данный тип активности устарел, так как не слишком удобен для планшетов. Теперь предпочтительнее использовать **ListFragment**. Но в основе всё равно лежит компонент **ListView** и базовые приёмы работы не изменились. Изучив данный пример, вы без труда разберётесь и с другими формами отображения списков.

Шаг первый

образом, что на экране есть только прокручиваемыи список и ему не нужна дополнительная разметка. Поэтому набираемся смелости, выбираем в папке **res/layout** файл **activity_main.xml** и удаляем его.

Шаг второй

Всё пропало! Теперь ничего не запустится! Don't panic! Открываем java-файл и видим, что студия ругается на строчку **setContentView(R.layout.activity_main);**, что вполне объяснимо. Мы ведь только что сами удалили файл разметки. Ещё раз набираемся смелости и удаляем эту строчку, она там тоже больше не нужна.

Шаг третий

Теперь нужно поставить Android в известность, что мы собираемся использовать экран со списком, поэтому меняем в строчке **public class ВашеНазваниеActivity extends AppCompatActivity** слово **AppCompatActivity** (или **Activity**) на **ListActivity**. Если набирать вручную, то студия автоматически импортирует нужный класс.

В результате в секции **import** нашего файла появится новая строка. Там же мы увидим строку, которую можно безболезненно удалить:

```
import android.app.ListActivity;
~~import android.app.Activity;~~
```

Шаг четвёртый

Подготовительные работы закончены. Теперь пришло время подготовить данные для списка, чтобы отобразить их на экране. Создадим массив строк:

```
final String[] catNamesArray = new String[] { "Рыжик", "Барсик", "Мурзик",
                "Мурка", "Васька", "Томасина", "Бобик", "Кристина", "Пушок",
                "Дымка", "Кузя", "Китти", "Барбос", "Масяня", "Симба" };
```

К слову сказать, вы можете создать массив строк в ресурсах, в этом случае вам будет проще редактировать список, не затрагивая код программы. Когда наберётесь опыта, то сами решите, какой вариант лучше.

Шаг пятый

А теперь начинается самое важное. У нас есть намерение создать экран со списком и сами слова для списка. Необходим некий посредник, который свяжет эти звенья в одно целое. Для подобных целей в Android существует понятие *адаптера данных* и его определение для работы с массивами строк выглядит так:

Адаптеру нужно от вас три вещи: ~~явки, пароли, деньги~~, текущий контекст, идентификатор ресурса с разметкой для каждой строки, массив строк.

Мы можем ему предложить **ListActivity** в качестве текущего контекста (можно использовать ключевое слово **this**), готовый системный идентификатор ресурса и созданный массив строк. А выглядеть это будет так:

```
private ArrayAdapter<String> mAdapter;

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

    mAdapter = new ArrayAdapter<>(this,
            android.R.layout.simple_list_item_1, catNamesArray);
}
```

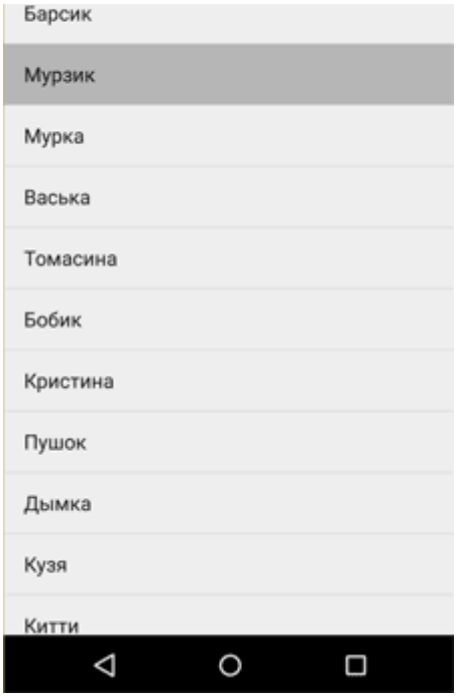
Обратите внимание на строчку **android.R.layout.simple_list_item_1**. В ней уже содержится необходимая разметка для отдельного элемента списка, которая состоит из одного компонента **TextView**. Если вас не устраивает системная разметка, то можете создать собственную разметку в xml-файле и подключить её. Об этом в следующий раз.

Шаг шестой

Осталось сделать заключительный штрих - подключить адаптер:

```
setListAdapter(mAdapter);
```

Запускаем проект и любуемся своим списком. Он прекрасно прокручивается и каждый пункт подсвечивается при нажатии.



Обработка нажатий

Но пока приложение никак не реагирует на наши нажатия. Исправим ситуацию. Нам нужно знать, на каком пункте списка осуществляется нажатие. У **ListActivity** есть специальный метод для таких случаев - **onListItemClick()**. Начинайте вводить первые символы названия метода и студия предложит вам подходящий вариант. Нажмите Enter на предложенном варианте и у вас появится заготовка.

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
}
```

У метода четыре параметра. Самым интересным является третий параметр **position**, который указывает на номер выбранного пункта списка.

Осталось только прописать код для события - давайте выведем всплывающее сообщение, которое будет содержать позицию выбранного элемента списка.

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Toast.makeText(getApplicationContext(),
            "Вы выбрали " + (position + 1) + " элемент", Toast.LENGTH_SHORT).show();
}
```

Замурчательно. Но хочется узнать не номер выбранного пункта, а сам текст. У списка **ListView** есть специальный метод **getItemAtPosition(position)**, возвращающий объект для заданной позиции. Перепишем код.

```
Toast.makeText(getApplicationContext(),
        "Вы выбрали " + l.getItemAtPosition(position).toString(),
Toast.LENGTH_SHORT).show();
```

В данном случае мы используем первый параметр **l**, который отвечает за родительский компонент **ListView**. Возвращаемый объект нужно преобразовать в строку.

В тех методах, у которых нет в параметрах ссылки на **ListView**, мы можем получить доступ к списку через метод активности **getListView()**.

Запускаем программу и начинаем щёлкать по любой позиции списка - мы получим соответствующее сообщение. Вы можете использовать свой код - вызывать новое окно, проигрывать музыку и т.д.

Долгое нажатие и удаление элемента списка

Расширим возможности списка и научимся обрабатывать долгие нажатия, а также удалять некоторые элементы списка.

Для долгого нажатия существует интерфейс **OnItemLongClickListener** с методом **onItemLongClick()**, возвращающим значение. Так как мы собираемся обрабатывать долгие нажатия, то строчку **return false;** необходимо заменить на **return true;**.

Добавляем интерфейс в активность, вручную вводя текст **implements OnItem**, студия предложит подсказку и поможет создать нужный метод для данного интерфейса.

```
public class MainActivity extends ListActivity implements
AdapterView.OnItemLongClickListener {
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, int position,
long id) {
        return true;
    }
}
```

Далее внесём небольшое изменение в адаптер данных. Сам по себе массив строк является неизменяемым, и чтобы мы могли удалять пункты из списка, необходимо сконвертировать его в специальный объект **ArrayList<String>**, который является изменяемым, а уже новый объект отдадим адаптеру. Объявим новую переменную.

```
(ArrayList(catNamesArray));
```

Подключаем к адаптеру.

```
mAdapter = new ArrayAdapter<>(this,
        android.R.layout.simple_list_item_1, catNamesList); //вместо catNamesArray
стало catNamesList
```

Далее прописываем необходимый код для удаления выбранного пункта меню и запускаем программу. Прокручивая список, с удивлением замечаем, что среди кошачьих имён затесался какой-то сраный пёсик Бобик. Пробуем удалить его. Получилось! Теперь наш список выглядит правильно.

```
@Override
public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long
id) {
    String selectedItem = parent.getItemAtPosition(position).toString();

    mAdapter.remove(selectedItem);
    mAdapter.notifyDataSetChanged();

    Toast.makeText(getApplicationContext(),
            selectedItem + " удалён.",
            Toast.LENGTH_SHORT).show();
    return true;
}
```

Метод **remove()** удаляет элемент из списочного массива, а метод **notifyDataSetChanged()** уведомляет список об изменении данных для обновления списка на экране.

На всякий случай ещё раз просмотрите список и если увидите чужеродное имя, то удалите его.

Удаление - весьма опасная операция, пользователь может по ошибке нажать на пункт списка. Лучшим решением было бы показать диалоговое окно с подтверждением операции. В последнее время весьма популярным стало использование специального типа уведомления внизу экрана с кнопкой "Отмена", например, готовый компонент **SnackBar** (о нём говорилось на одном из уроков).

Заключение

Поначалу эта статья может показаться вам сложной. Не отчаивайтесь, возьмите её как шаблон и на первых порах просто копируйте куски кода. Позже с практикой вы лучше разберётесь в работе со списком.

статью про элемент управления ListView, а также статью Списки со значками.

Исходный код

▶ Показать код (щелкните лапкой)

Своя разметка

Когда в самом начале статьи я говорил, что для **ListActivity** не нужен шаблон **activity_main.xml**, то немножко лукавил. На самом деле вы можете подключить свой шаблон, но с одним условием - шаблон должен содержать элемент **ListView** с идентификатором **@android:id/list**.

Можно заново создать файл **activity_main.xml**, если вы его удалили, как вас просили, или файл с другим именем, например, **activity_customlist.xml**:

```
<?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="match_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@android:id/list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#00FF00" />

    <TextView
        android:id="@android:id/empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="List is Empty" />

</LinearLayout>
```

Я специально установил зелёный цвет для фона, чтобы вы поверили, что будет запускаться наш шаблон вместо системного, а **TextView** с системным идентификатором **android:id/empty** нужен для отображения текста, если список будет пустым. Осталось добавить строчку кода, который подключает шаблон:

```
setContentView(R.layout.activity_customlist);
```

Запустите проект и убедитесь, что загружается наш шаблон. Если вы зададите пустой массив, то вместо списка вы увидите **TextView** с текстом *List is Empty*.

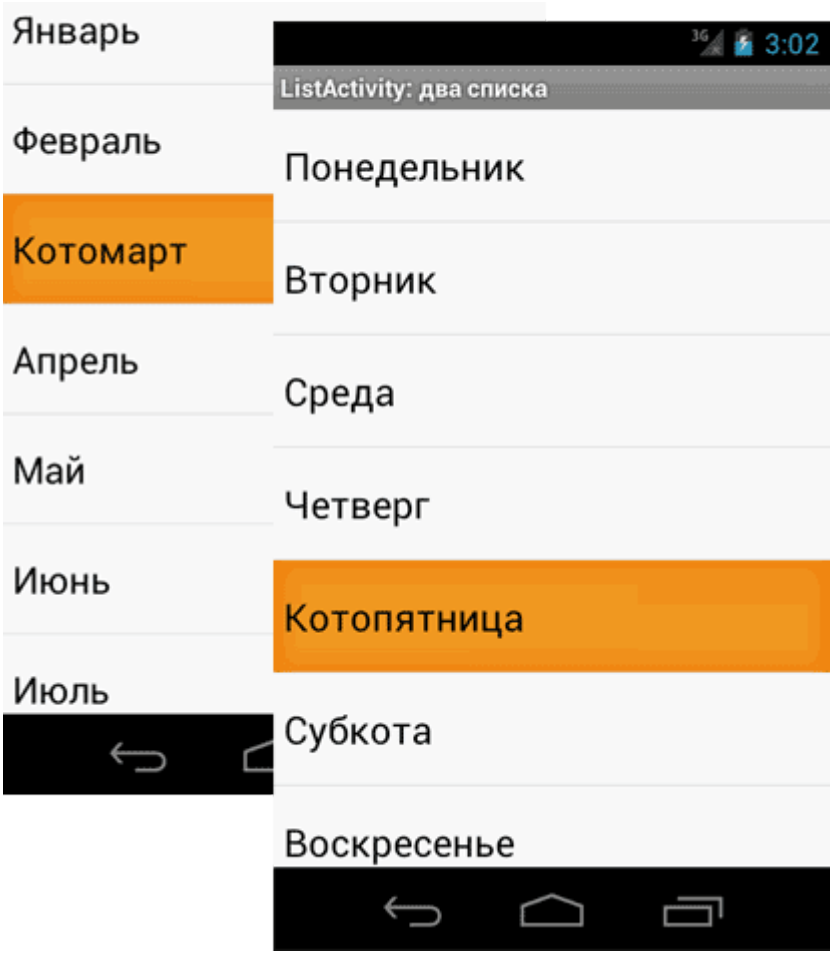
надо изучить поближе **ListView**. Для этого на сайте есть отдельный раздел.

Переключаемся между двумя списками

Возможно, вам понадобится переходить из одного списка в другой. Например, первый список представляет собой месяцы, а второй - дни недели.

```
    private String[] mMonthArray = { "Январь", "Февраль", "Котомарт", "Апрель", "Май",
            "Июнь", "Июль", "Август", "Сентябрь", "Октябрь", "Ноябрь",
            "Декабрь" };

    private String[] mDayOfWeekArray = new String[] { "Понедельник", "Вторник",
"Среда",
            "Четверг", "Котопятница", "Субкота", "Воскресенье" };

    // Создадим два адаптера
    private ArrayAdapter<String> mMonthAdapter, mWeekOfDayAdapter;
    private String mMonth, mDayOfWeek;

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

        mMonthAdapter = new ArrayAdapter<>(this,
                android.R.layout.simple_list_item_1, mMonthArray);
        mWeekOfDayAdapter = new ArrayAdapter<>(this,
                android.R.layout.simple_list_item_1, mDayOfWeekArray);

        setListAdapter(mMonthAdapter);
    }

    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
        super.onListItemClick(l, v, position, id);
        if (getListAdapter() == mMonthAdapter) {
            mMonth = (String) l.getItemAtPosition(position);
            setListAdapter(mWeekOfDayAdapter);
            mWeekOfDayAdapter.notifyDataSetChanged();
        } else {
            mDayOfWeek = (String) l.getItemAtPosition(position);
            Toast.makeText(getBaseContext(), mMonth + ": " + mDayOfWeek,
                    Toast.LENGTH_LONG).show();
            setListAdapter(mMonthAdapter);
            mMonthAdapter.notifyDataSetChanged();
        }
    }
}
```

Мы создали два адаптера через массивы строк. Сначала используем первый адаптер. При выборе элемента списка через метод **onListItemClick()** подключаем другой адаптер. Чтобы изменения отразились на экране, необходимо вызвать метод **notifyDataSetChanged()**.



Дополнительное чтение

Обсуждение статьи на форуме.