MultiType ListView 사용법

목차

1. 1. 아이템이 다른 ListView

두 종류 이상의 View 를 ListView 의 아이템으로 사용하는 방법에 대해서 알아 보도록 하겠습니다.

Adapter 에서 거의 대부분의 작업이 이루어지며, 몇 개의 함수를 새로 만들거나 조금 수정해주면 됩니다. 물론, 새로운 View 에 대한 Layout 리소스는 추가해야겠지요.

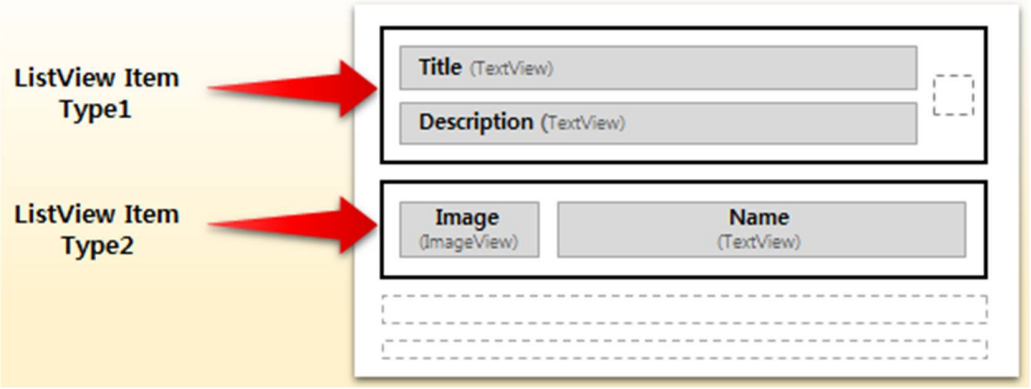
구체적인 작업 순서를 언급하자면 아래와 같습니다.

- ✓ 새로운 아이템 View 에 대한 Layout 리소스 추가.
- ✓ Adatper 에서 제공할 View 타입에 대한 상수 정의.
- ✓ View 타입과 관련하여 Adapter 에서 제공하는 함수 오버라이딩.
- ✓ getViewTypeCount() : View 타입에 대한 갯수 리턴
- ✓ getItemViewType() : position 에 해당하는 View 타입 리턴.
- ✓ Adapter 의 getView() 내용 수정.
- ✓ 현재 position 에 따른 View 생성.

2. ListView 의 아이템으로 두 가지 이상의 View 사용하기

아이템이 다른 ListView 를 만드는 방법을 이해하기 위해서는 커스텀 ListView 만드는 방법에 대해 숙지하고 있어야 합니다.

이 예제에서는 두 개의 View 를 하나의 ListView 아이템으로 사용합니다. 각 View 는 아래와 같이 구성됩니다.



2.1 activity_main.xml 에 ListView 추가

ListView 가 표시될 위치를 결정하여 ListView 를 추가합니다. 예제에서는 MainActivity 에 ListView 를 추가하므로 "activity_main.xml" 파일(또는 "content_main.xml")에 관련 코드를 작성합니다. 아이템 간 영역 구분을 좀 더 명확하게 표시하기 위해 "divider"를 사용하였습니다.

- "activity_main.xml" - MainActivity 에 ListView 추가

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context="com.examples.listviewdifferentitemexample1.MainActivity"
    tools:showIn="@layout/activity_main">

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listview1"
        android:divider="#000000"
        android:dividerHeight="5dp"/>
</RelativeLayout>
```

2.2 ListView 아이템 뷰 만들기

아이템에 사용되는 두 개의 View 타입에 따라 각각 Layout 리소스 XML 을 작성합니다.

2.2.1 "listview_item1.xml" 만들기

- "listview_item1.xml" – 첫 번째 아이템에 대한 Layout 리소스 XML 정의

```
<?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="match_parent">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="9">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/title"
            android:textSize="28sp"/>
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/desc"
            android:textSize="16sp"/>
    </LinearLayout>
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="1"/>
</LinearLayout>
```

다음은 ImageView 와 TextView 로 이루어진 아이템에 대한 Layout 리소스 XML 파일입니다.

2.2.2 "listview_item2.xml" 만들기

"listview_item2.xml" - 두 번째 아이템에 대한 Layout 리소스 XML 정의

```
<?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="match_parent">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/image"
        android:layout_weight="1"
        android:scaleType="center"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:text="Name"
        android:id="@+id/name"
        android:textSize="20sp"
        android:textColor="#000000"
        android:gravity="center_vertical"
        android:layout_weight="4" />
</LinearLayout>
```

2.3 모델 클래스 만들기

ListView 아이템에 대한 View 가 두 종류라 하더라도 각각에 대해 클래스를 정의할 필요는 없습니다. 두 가지 타입의 View 에 표시될 데이터를 모두 포함하는 하나의 클래스만 정의한 다음, View 타입에 따라 필요한 멤버만 사용하면 되는거죠.

물론 객체지향 개념에 충실하게 아이템을 추상화하는 부모 클래스를 정의하고 각 View 타입에 필요한 멤버를 가지는 두 개의 자식 클래스를 상속한 다음, 아이템이 어떠한 클래스에 대한 객체를 가지고 있는지를 판단하여 객체에 대한 레퍼런스를 사용하는 방법도 있습니다. Adapter 에서는 부모 클래스에 대한 리스트만 유지하구요.

아니면 별도로 두 개의 클래스를 정의하고 Adapter 에서 두 가지 클래스 객체에 대한 리스트를 유지하는 방법을 사용할 수도 있겠죠.

일단 아이템 데이터 클래스 구현은 본인의 스타일에 따라 편한대로 선택하면 될 것 같습니다. 세 가지 방법 모두 장 단점이 있는 방법이니까요.

살짝 다른 설명이 길었군요. 이제 첫 번째 설명한 방법대로 View 들의 모든 위젯 데이터를 포함하는 아이템에 대한 클래스를 정의하겠습니다. "ModelItem.java" 파일을 생성합니다.

- "ModelItem.java" - 모델 클래스 정의

```
package com.example.actor.simplemultitype;

import android.graphics.drawable.Drawable;

public class ModelItem {
    // View type을 위한 필드: 1 or 2
    private int type ;

    // listview_item1.xml 의 데이터
    private String title ; // TextView
    private String desc  ; // TextView

    // listview_item2.xml 의 데이터
    private Drawable image ; // ImageView
    private String   name  ; // TextView

    // getter & setter
    public int getType() {
        return type;
    }
```

```
    public void setType(int type) {
        this.type = type;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    public Drawable getImage() {
        return image;
    }

    public void setImage(Drawable image) {
        this.image = image;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // toString


    @Override
    public String toString() {
        return "ModelItem{" +
                "type=" + type +
                ", title='" + title + '\'' +
```

```
                ", desc='" + desc + '\'' +
                ", image=" + image +
                ", name='" + name + '\'' +
                '}';
    }

    // contsturtor
    public ModelItem() {
    }

    // type 1 생성자
    public ModelItem(String title, String desc) {
        this.type  = AdapterItem.ITEM_VIEW_TYPE_TEXT;
        this.title = title;
        this.desc  = desc;
    }

    // type 2 생성자
    public ModelItem(String name, Drawable image) {
        this.type  = AdapterItem.ITEM_VIEW_TYPE_IMAGES;
        this.image = image;
        this.name  = name;
    }
}
```

2.4 Adapter 구현

Adapter 에 몇 가지 추가 작업을 해주면 다중 아이템 지원을 위한 Adapter 를 구현할 수 있습니다.

- Adapter 구현 순서

1. 아이템 View 타입에 대한 상수를 정의
2. **getViewTypeCount()** 오버라이딩.
3. **getItemViewType()** 오버라이딩.
4. **getView()** 오버라이딩

- "ListViewAdapter.java" - BaseAdapter 상속 및 ListViewAdapter 구현.

```
public class AdapterItem extends ArrayAdapter<ModelItem> {

    // ViewType 상수선언
    public static final int ITEM_VIEW_TYPE_TEXT   = 0;
    public static final int ITEM_VIEW_TYPE_IMAGES = 1;

    private final Context context;
    private final List<ModelItem> datas;
    private final LayoutInflater inflater;

    public AdapterItem(@NonNull Context context, int resource, @NonNull List<ModelItem>
objects) {
        super(context, resource, objects);
        this.context = context;
        this.datas   = objects;
        this.inflater =  LayoutInflater.from(context);
    }

    // 필수 구현
    @Override
    public int getItemViewType(int position) {
        return datas.get(position).getType();
    }

    // 필수 구현 : position 위치의 아이템 타입 리턴.
    @Override
    public int getViewTypeCount() {
        return 2;
    }
```

```
    private class ViewHolder {
        // type 1
        TextView title ;
        TextView desc  ;

        // type 2
        ImageView image;
        TextView  name ;
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup
parent) {
        View rowView  = convertView;

        ModelItem item = datas.get( position );
        int viewType = getItemViewType(position);
        Log.d("viewType", viewType + "////" + item.toString() );

        ViewHolder holder = null;


        // view 가져오기.
        if( rowView == null ){
            // inflation

            holder = new ViewHolder();

            switch ( viewType ) {
                case ITEM_VIEW_TYPE_TEXT:
                    rowView = inflater.inflate(R.layout.listview_item1, parent, false);

                    holder.title = rowView.findViewById(R.id.title);
                    holder.desc  = rowView.findViewById(R.id.desc );
                    break;
                case ITEM_VIEW_TYPE_IMAGES:
                    rowView = inflater.inflate(R.layout.listview_item2, parent, false);

                    holder.image = rowView.findViewById(R.id.image);
                    holder.name  = rowView.findViewById(R.id.name);
                    break;
            }
```

```
            rowView.setTag( holder );
        }
        else {
            holder = (ViewHolder) rowView.getTag();
        }

        // 데이터 넣기
        switch ( viewType ) {
            case ITEM_VIEW_TYPE_TEXT:
                holder.title.setText( item.getTitle() );
                holder.desc.setText( item.getDesc() );
                break;
            case ITEM_VIEW_TYPE_IMAGES:
                holder.image.setImageDrawable( item.getImage() );
                holder.name.setText( item.getName() );
                break;
        }

        return  rowView;

    }
}
```

2.5 Adapter 생성 및 데이터 추가

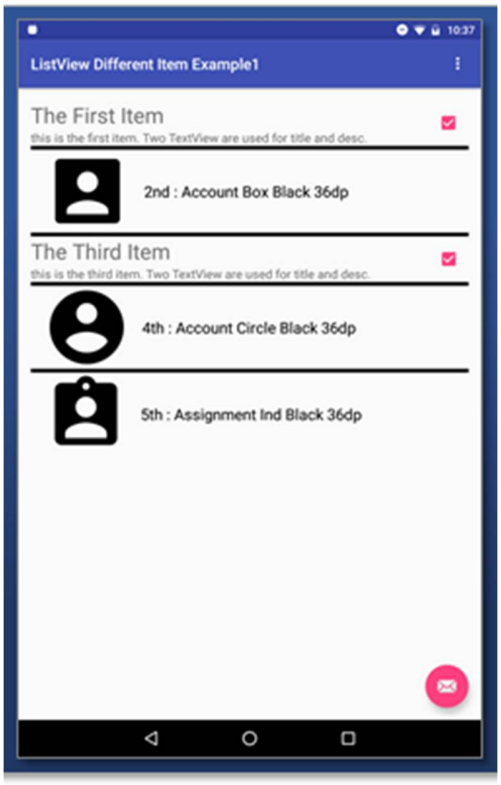
- "MainActivity.java" - onCreate() 함수에서 ListView 및 Adapter 생성.

```
public class MainActivity extends AppCompatActivity {

    // 위젯 선언
    private ListView listview1;
    private List<ModelItem> list;
    private AdapterItem adapter ;

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

        // 위젯 찾기
        listview1 = findViewById(R.id.listview1);

        // 위젯 설정: 리스너 설정. 없음.

        // 데이터 만들기
        list = makeData();

        // adapter 만들기
        this.adapter = new AdapterItem(MainActivity.this, -1, list);

        // listview 와 adapter 연결하기.
        listview1.setAdapter( this.adapter );
    }

    private List<ModelItem> makeData() {
        List<ModelItem> list = new ArrayList<>();
        list.add( new ModelItem( "title1 " ," desc1 "));
        list.add( new ModelItem( "name 1",
getResources().getDrawable(R.drawable.sample_1)));
        list.add( new ModelItem( "name 2",
getResources().getDrawable(R.drawable.sample_2)));
        list.add( new ModelItem( "title2 " ," desc2 "));
        list.add( new ModelItem( "name 3",
getResources().getDrawable(R.drawable.sample_3)));
        list.add( new ModelItem( "title3 " ," desc3 "));
        list.add( new ModelItem( "name 4",
getResources().getDrawable(R.drawable.sample_4)));
        list.add( new ModelItem( "name 5",
getResources().getDrawable(R.drawable.sample_5)));
        list.add( new ModelItem( "title4 " ," desc4 "));
```

```
        list.add( new ModelItem( "title5 " ," desc5 "));

        return list;
    }
}
```

3. 아이템이 다른 ListView 예제 실행 화면

예제를 순서대로 작성하고 실행하면 아래와 같은 화면이 출력되며 다른 형태의 View 가 ListView 아이템에 표시됨을 확인할 수 있습니다.



4. Reference

http://recipes4dev.tistory.com/57