RecyclerView

RecyclerView 是 android-support-v7-21 版本中新增的一个 Widgets。它所提供的功能于ListView、GridView基本相同，就是显示大量的**同质数据**。

以下是它的新特性：

分离出数据显示相关的所有问题，只关注View的回收和复用

对数据的显示、动画的配置进行高度解耦，提供了插拔式的体验

强制使用ViewHolder模式，消除不必要的Layout Resource查找时间

内置添加和移除动画

1.导入类库

> compile ``'com.android.support:recyclerview-v7:+'

2.相关类介绍

RecyclerView.Adapter：包装数据集合并为每个条目创建视图

RecyclerView.ViewHolder：保存用于显示数据的子View

LayoutManager：控制条目视图放置的位置、方式

ItemDecoration：绘制装饰视图

ItemAnimator：ItemView的动画效果

3.生命周期

getItemViewType：获取View类型

onCreateViewHolder：创建ViewHolder

onViewAttachedToWindow：当Item可见时调用

onBindViewHolder：绑定数据以及逻辑控制样式

onViewDetachedFromWindow：Item不可见时调用

onViewRecycled：Item被回收时调用

4.刷新方法

notifyDataSetChanged：刷新所有

swapAdapter：内部会调用setAdapter（），如果ViewHolder相同，不会清空RecycledViewPool

notifyItemChanged：顾名思义，更新部分指定数据，更加灵活

5.关于onItemClick（）

刚开始使用的时候竟然发现不能设置onItemClickListener，这也是之前ListView耦合度太高才有的特性。那么我们可以：在onBindViewHolder或ViewHolder初始化中实现，定义Interface进行回调。

用法示例：

创建

**recyclerView** = **new** RecyclerView(**this**);  
 setContentView(**recyclerView**);

设置布局  
 **recyclerView**.setLayoutManager(**new** GridLayoutManager(**this**,3)); *//三列  
或recyclerView.setLayoutManager(new LinearLayoutManager(this,LinearLayoutManager.HORIZONTAL,false));*

*设置适配器* **recyclerView**.setAdapter(**new** MyAdapter());

适配器类：

**class** ViewHolder **extends** RecyclerView.ViewHolder {  
 **private** View **root**;  
 **private** TextView **textTitle**,**textContent**;  
  
 **public** ViewHolder(View root) {  
 **super**(root);  
 **textTitle** = (TextView) root.findViewById(R.id.***textTitle***);  
 **textContent** = (TextView) root.findViewById(R.id.***textContent***);  
 }  
  
 **public** TextView getTextContent() {  
 **return textContent**;  
 }  
  
 **public** TextView getTextTitle() {  
 **return textTitle**;  
 }  
 }  
  
 @Override  
 **public** RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, **int** viewType) {  
 **return new** ViewHolder(LayoutInflater.*from*(parent.getContext()).inflate(R.layout.***cell\_list***,**null**));  
 }  
  
 @Override  
 **public void** onBindViewHolder(RecyclerView.ViewHolder holder, **int** position) {  
 ViewHolder viewHolder = (ViewHolder) holder;  
 CellData cellData = **data**[position];  
 viewHolder.getTextTitle().setText(cellData.**title**);  
 viewHolder.getTextContent().setText(cellData.**content**);  
  
 }  
  
 @Override  
 **public int** getItemCount() { *//h获取子对象的数量* **return data**.**length**;  
 }  
  
  
}

**ListView**

列表的显示需要三个元素

1．ListVeiw 用来展示列表的View。

2．适配器 用来把数据映射到ListView上的中介。

3．数据 具体的将被映射的字符串，图片，或者基本组件。

根据列表的适配器类型，列表分为三种，ArrayAdapter，SimpleAdapter和SimpleCursorAdapter

其中以ArrayAdapter最为简单，只能展示一行字。SimpleAdapter有最好的扩充性，可以自定义出各种效果。SimpleCursorAdapter可以认为是SimpleAdapter对数据库的简单结合，可以方面的**把数据库的内容以列表的形式展示出来**。

1. ArrayAdapter

public class MyListView extends Activity {

private ListView listView;

@Override

public void onCreate(Bundle savedInstanceState){

super.onCreate(savedInstanceState);

listView = new ListView(this);

listView.setAdapter(new ArrayAdapter<String>

(this, android.R.layout.simple\_expandable\_list\_item\_1,getData()));

setContentView(listView);

}

private List<String> getData(){

List<String> data = new ArrayList<String>();

data.add("测试数据1");

data.add("测试数据2");

data.add("测试数据3");

data.add("测试数据4");

return data;

}

}

上面代码使用了ArrayAdapter(Context context, int textViewResourceId, List<T> objects)来装配数据，要装配这些数据就需要一个连接ListView视图对象和数组数据的适配器来两者的适配工作，ArrayAdapter的构造需要三个参数，依次为this,布局文件（注意这里的布局文件描述的是列表的每一行的布局，android.R.layout.simple\_list\_item\_1是系统定义好的**布局文件只显示一行文字**，数据源(一个List集合)。同时用setAdapter（）完成适配的最后工作。

2. SimpleCursorAdapter

简单的说就是方便把从游标得到的数据进行列表显示，并可以把指定的列映射到对应的TextView中。

下面的程序是从电话簿中把联系人显示到类表中。先在通讯录中添加一个联系人作为数据库的数据。然后获得一个指向数据库的Cursor并且定义一个布局文件（当然也可以使用系统自带的）。

public class MyListView2 extends Activity {

private ListView listView;

public void onCreate(Bundle savedInstanceState){

super.onCreate(savedInstanceState);

listView = new ListView(this);

//先获得一个指向系统通讯录数据库的Cursor对象获得数据来源

Cursor cursor = getContentResolver().query(People.CONTENT\_URI, null, null, null, null);

//将获得的Cursor对象交由Activity管理，这样Cursor的生命周期和Activity便能够自动同步

startManagingCursor(cursor);

listView.setAdapter(new

SimpleCursorAdapter(this, android.R.layout.simple\_expandable\_list\_item\_1,

cursor, new String[]{People.NAME}, new int[]{android.R.id.text1});

);

setContentView(listView);

}

}

最后两个参数：一个包含数据库的列的String型数组，一个包含布局文件中对应组件id的int型数组。其作用是自动的将String型数组所表示的每一列数据映射到布局文件对应id的组件上。上面的代码，将NAME列的数据一次映射到布局文件的id为text1的组件上。

注意：需要在AndroidManifest.xml中如权限：<uses-permission android:name="android.permission.READ\_CONTACTS"></uses-permission>

3. SimpleAdapter

**simpleAdapter的扩展性最好**，可以定义各种各样的布局出来，可以放上ImageView，还可以放上Button，CheckBox（复选框）等等。

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="horizontal" android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent">

<ImageView android:id="@+id/img"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_margin="5px"/>

<LinearLayout android:orientation="vertical"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

<TextView android:id="@+id/title"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="#FFFFFFFF"

android:textSize="22px" />

<TextView android:id="@+id/info"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="#FFFFFFFF"

android:textSize="13px" />

</LinearLayout>

</LinearLayout>

下面是实现代码：

public class MyListView3 extends ListActivity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

listview= new ListView(this);

SimpleAdapter adapter = new SimpleAdapter(this,getData(),R.layout.main,

new String[]{"title","info","img"},

new int[]{R.id.title,R.id.info,R.id.img});

listview.setAdapter(adapter);

setContentView(listview);

}

private List<Map<String, Object>> getData() {

List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();

Map<String, Object> map = new HashMap<String, Object>();

map.put("title", "G1");

map.put("info", "google 1");

map.put("img", R.drawable.i1);

list.add(map);

map = new HashMap<String, Object>();

map.put("title", "G2");

map.put("info", "google 2");

map.put("img", R.drawable.i2);

list.add(map);

map = new HashMap<String, Object>();

map.put("title", "G3");

map.put("info", "google 3");

map.put("img", R.drawable.i3);

list.add(map);

return list;

}

}

**使用simpleAdapter的数据用一般都是HashMap构成的List**，list的每一节对应ListView的每一行。HashMap的每个键值数据映射到布局文件中对应id的组件上。下面做适配，new一个SimpleAdapter参数一次是：this，布局文件（main.xml），HashMap的 title 和 info，img。布局文件的组件id，title，info，img。布局文件的各组件分别映射到HashMap的各元素上，完成适配。

有按钮的ListView

因为按钮是无法映射的，即使你成功的用布局文件显示出了按钮也无法添加按钮的响应，这时就要研究一下ListView是如何现实的了，而且必须要重写一个类继承BaseAdapter。下面的示例将显示一个按钮和一个图片，两行字如果单击按钮将删除此按钮的所在行。

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="horizontal"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent">

<ImageView android:id="@+id/img"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_margin="5px"/>

<LinearLayout android:orientation="vertical"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content">

<TextView android:id="@+id/title"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="#FFFFFFFF"

android:textSize="22px" />

<TextView android:id="@+id/info"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textColor="#FFFFFFFF"

android:textSize="13px" />

</LinearLayout>

<Button android:id="@+id/view\_btn"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="介绍"

android:layout\_gravity="bottom|right" />

</LinearLayout>

程序代码：

public class MyListView4 extends Activity {

private List<Map<String, Object>> mData;

private ListView listView;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

listView=new ListView(this);

mData = getData();

MyAdapter adapter = new MyAdapter(this);

listView.setAdapter(adapter);

setContentView(listView); }

private List<Map<String, Object>> getData() {

List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();

Map<String, Object> map = new HashMap<String, Object>();

map.put("title", "G1");

map.put("info", "google 1");

map.put("img", R.drawable.i1);

list.add(map);

map = new HashMap<String, Object>();

map.put("title", "G2");

map.put("info", "google 2");

map.put("img", R.drawable.i2);

list.add(map);

map = new HashMap<String, Object>();

map.put("title", "G3");

map.put("info", "google 3");

map.put("img", R.drawable.i3);

list.add(map);

return list;

}

// ListView 中某项被选中后的逻辑

@Override

protected void onListItemClick(ListView l, View v, int position, long id) {

Log.v("MyListView4-click", (String)mData.get(position).get("title"));

}

/\*\*

\* listview中点击按键弹出对话框

\*/

public void showInfo(){

new AlertDialog.Builder(this)

.setTitle("我的listview")

.setMessage("介绍...")

.setPositiveButton("确定", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

}

})

.show();

}

public final class ViewHolder{

public ImageView img;

public TextView title;

public TextView info;

public Button viewBtn;

}

public class MyAdapter extends BaseAdapter{

private LayoutInflater mInflater;

public MyAdapter(Context context){

this.mInflater = LayoutInflater.from(context);

}

@Override

public int getCount() {

// TODO Auto-generated method stub

return mData.size();

}

@Override

public Object getItem(int arg0) {

// TODO Auto-generated method stub

return null;

}

@Override

public long getItemId(int arg0) {

// TODO Auto-generated method stub

return 0;

}

@Override

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

ViewHolder holder = null;

if (convertView == null) {

holder=new ViewHolder();

convertView = mInflater.inflate(R.layout.vlist2, null);

holder.img = (ImageView)convertView.findViewById(R.id.img);

holder.title = (TextView)convertView.findViewById(R.id.title);

holder.info = (TextView)convertView.findViewById(R.id.info);

holder.viewBtn = (Button)convertView.findViewById(R.id.view\_btn);

convertView.setTag(holder);

}else {

holder = (ViewHolder)convertView.getTag();

}

holder.img.setBackgroundResource((Integer)mData.get(position).get("img"));

holder.title.setText((String)mData.get(position).get("title"));

holder.info.setText((String)mData.get(position).get("info"));

holder.viewBtn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

showInfo();

}

});

return convertView;

}

}

}

　　下面将对上述代码，做详细的解释，listView在开始绘制的时候，系统首先调用getCount（）函数，根据他的返回值得到listView的长度（这也是为什么在开始的第一张图特别的标出列表长度），然后根据这个长度，调用getView（）逐一绘制每一行。如果你的getCount（）返回值是0的话，列表将不显示；同样return 1，就只显示一行。

系统显示列表时，首先实例化一个适配器（这里将实例化自定义的适配器）。当手动完成适配时，必须手动映射数据，这需要重写getView（）方法。系统在绘制列表的每一行的时候将调用此方法。getView()有三个参数，position表示将显示的是第几行，covertView是从布局文件中inflate来的布局。我们用LayoutInflater的方法将定义好的xml文件提取成View实例用来显示。然后将xml文件中的各个组件实例化（简单的findViewById()方法）。这样便可以将数据对应到各个组件上了。但是按钮为了响应点击事件，需要为它添加点击监听器，这样就能捕获点击事件。至此一个自定义的listView就完成了，现在让我们回过头从新审视这个过程。系统要绘制ListView了，他首先获得要绘制的这个列表的长度，然后开始绘制第一行，怎么绘制呢？调用getView()函数。在这个函数里面首先获得一个View（实际上是一个ViewGroup），然后再实例并设置各个组件，显示之。好了，绘制完这一行了。那再绘制下一行，直到绘完为止。在实际的运行过程中会发现listView的每一行没有焦点了，这是因为Button抢夺了listView的焦点，只要布局文件中将Button设置为没有焦点就OK了。

**AlertDialog**

AlertDialog.Builder dialog = new AlertDialog.Builder (MainActivity.this); //创建实例

dialog.setTitle("This is Dialog");

dialog.setMessage("Something important.");

dialog.setCancelable(false); //按返回键不能退出

dialog.setPositiveButton("OK", new DialogInterface.OnClickListener() { //设置确定按钮点击事件

@Override

public void onClick(DialogInterface dialog, int which) {

}

});

dialog.setNegativeButton("Cancel", new DialogInterface.OnClickListener() { //设置取消按钮点击事件

@Override

public void onClick(DialogInterface dialog, int which) {

}

});

dialog.show(); //将对话框显示出来