[Android开发之RecyclerView的使用全解](http://blog.csdn.net/dmk877/article/details/50816933)

通过本篇博客，你将学到以下知识点

①RecyclerView与ListView相比它的优点

②RecyclerView的初步用法

③RecyclerView增加分隔线

④RecyclerView更改分隔线的样式

⑤RecyclerView的Adapter的用法

⑥RecyclerView.Adapter中刷新的几个方法的对比

⑦给RecyclerView增加条目点击事件

**1.RecyclerView是什么？**

      RecylerView是support-v7包中的新组件，是一个强大的滑动组件，与经典的ListView相比，同样拥有item回收复用的功能，这一点从它的名字recylerview即回收view也可以看出。看到这也许有人会问，不是已经有ListView了吗，为什么还要RecylerView呢？这就牵扯到第二个问题了。

**2.RecyclerView的优点是什么？**

     根据官方的介绍RecylerView是ListView的升级版，既然如此那RecylerView必然有它的优点，现就RecylerView相对于ListView的优点罗列如下：

①RecylerView封装了viewholder的回收复用，也就是说RecylerView标准化了ViewHolder，编写Adapter面向的是ViewHolder而不再是View了，复用的逻辑被封装了，写起来更加简单。

②提供了一种插拔式的体验，高度的解耦，异常的灵活，针对一个Item的显示RecylerView专门抽取出了相应的类，来控制Item的显示，使其的扩展性非常强。例如：你想控制横向或者纵向滑动列表效果可以通过LinearLayoutManager这个类来进行控制(与GridView效果对应的是GridLayoutManager,与瀑布流对应的还有StaggeredGridLayoutManager等)，也就是说RecylerView不再拘泥于ListView的线性展示方式，它也可以实现GridView的效果等多种效果。你想控制Item的分隔线，可以通过继承RecylerView的ItemDecoration这个类，然后针对自己的业务需求去抒写代码。

③可以控制Item增删的动画，可以通过ItemAnimator这个类进行控制，当然针对增删的动画，RecylerView有其自己默认的实现。

**3.RecyclerView的用法**

**3.1 RecyclerView的初步用法(包括RecyclerView.Adapter用法)**

     说了这么多，可能大家最关心的就是RecylerView应该怎么用，我们先来讨论讨论RecylerView的用法的理论知识，然后结合一个实例来体验一下RecylerView的优势，首先我们需要明白的一点是使用RecylerView必须导入support-v7包，在上面我提到过RecylerView高度的解耦，异常的灵活谷歌给我们提供了多个类来控制Item的显示。

recyclerView = (RecyclerView) findViewById(R.id.recyclerView);

LinearLayoutManager layoutManager = **new** LinearLayoutManager(**this** );

//设置布局管理器

recyclerView.setLayoutManager(layoutManager);

//设置为垂直布局，这也是默认的

layoutManager.setOrientation(OrientationHelper. VERTICAL);

//设置Adapter

recyclerView.setAdapter( recycleAdapter);

 //设置分隔线

recyclerView.addItemDecoration( **new** DividerGridItemDecoration(**this** ));

//设置增加或删除条目的动画

recyclerView.setItemAnimator( **new** DefaultItemAnimator());

可以看到对RecylerView的设置过程，比ListView要复杂一些，这也是RecylerView高度解耦的表现，虽然代码抒写上有点复杂，但它的扩展性是极高的。

在了解了RecyclerView的一些控制之后，紧接着来看看它的Adapter的写法，RecyclerView的Adapter与ListView的Adapter还是有点区别的，RecyclerView.Adapter，需要实现3个方法：

①onCreateViewHolder()

    这个方法主要生成为每个Item inflater出一个View，但是该方法返回的是一个ViewHolder。该方法把View直接封装在ViewHolder中，然后我们面向的是ViewHolder这个实例，当然这个ViewHolder需要我们自己去编写。直接省去了当初的convertView.setTag(holder)和convertView.getTag()这些繁琐的步骤。

②onBindViewHolder()

     这个方法主要用于适配渲染数据到View中。方法提供给你了一个viewHolder，而不是原来的convertView。

③getItemCount()

      这个方法就类似于BaseAdapter的getCount方法了，即总共有多少个条目。

实例：接着来几个小的实例帮助大家更深入的了解RecyclerView的用法，首先来实现一个最简单的列表



这种效果的MainAcitivity的代码如下

**package** com.example.reclerviewpractice;

**import** java.util.ArrayList;

**import** java.util.List;

**import** com.example.reclerviewpractice.adapter.MyRecyclerAdapter;

**import** android.annotation.SuppressLint;

**import** android.os.Bundle;

**import** android.support.v7.app.ActionBarActivity;

**import** android.support.v7.widget.DefaultItemAnimator;

**import** android.support.v7.widget.LinearLayoutManager;

**import** android.support.v7.widget.OrientationHelper;

**import** android.support.v7.widget.RecyclerView;

**public** **class** MainActivity **extends** ActionBarActivity {

**private** RecyclerView recyclerView;

**private** List<String> mDatas;

**private** MyRecyclerAdapter recycleAdapter;

     @SuppressLint("NewApi") @Override

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

           setContentView(R.layout.activity\_main);

            recyclerView = (RecyclerView) findViewById(R.id.recyclerView );

           initData();

            recycleAdapter= **new** MyRecyclerAdapter(MainActivity.**this** , mDatas );

           LinearLayoutManager layoutManager = **new** LinearLayoutManager(**this**);

            //设置布局管理器

            recyclerView.setLayoutManager(layoutManager);

            //设置为垂直布局，这也是默认的

           layoutManager.setOrientation(OrientationHelper.VERTICAL);

            //设置Adapter

            recyclerView.setAdapter( recycleAdapter);

            //设置增加或删除条目的动画

            recyclerView.setItemAnimator(**new** DefaultItemAnimator());

     }

**private** **void** initData() {

            mDatas = **new** ArrayList<String>();

**for** ( **int** i=0; i < 40; i++) {

                 mDatas.add( "item"+i);

           }

     }

}

RecyclerView的Adapter的代码如下:

**package** com.example.reclerviewpractice.adapter;

**import** java.util.List;

**import** com.example.reclerviewpractice.R;

**import** android.content.Context;

**import** android.support.v7.widget.RecyclerView;

**import** android.support.v7.widget.RecyclerView.ViewHolder;

**import** android.view.LayoutInflater;

**import** android.view.View;

**import** android.view.ViewGroup;

**import** android.widget.TextView;

**public** **class** MyRecyclerAdapter **extends** RecyclerView.Adapter<MyRecyclerAdapter.MyViewHolder> {

**private** List<String> mDatas;

**private** Context mContext;

**private** LayoutInflater inflater;

**public** MyRecyclerAdapter(Context context, List<String> datas){

**this**. mContext=context;

**this**. mDatas=datas;

            inflater=LayoutInflater. from(mContext);

     }

     @Override

**public** **int** getItemCount() {

**return** mDatas.size();

     }

     //填充onCreateViewHolder方法返回的holder中的控件

     @Override

**public** **void** onBindViewHolder(MyViewHolder holder, **final** **int** position) {

           holder.tv.setText( mDatas.get(position));

     }

     //重写onCreateViewHolder方法，返回一个自定义的ViewHolder

     @Override

**public** MyViewHolder onCreateViewHolder(ViewGroup parent, **int** viewType) {

           View view = inflater.inflate(R.layout.item\_home,parent, **false**);

           MyViewHolder holder= **new** MyViewHolder(view);

**return** holder;

     }

**class** MyViewHolder **extends** ViewHolder{

           TextView tv;

**public** MyViewHolder(View view) {

**super**(view);

                 tv=(TextView) view.findViewById(R.id. tv\_item);

           }

     }

}

可以看到RecyclerView标准化了ViewHolder，编写 Adapter面向的是ViewHoder而不在是View了，复用的逻辑被封装了，写起来更加简单。其实它的写法与BaseAdapter的写法是差不多的，大家可以对比下它与getView方法写法的区别，在onCreateViewHolder方法中初始化了一个View，然后返回一个ViewHolder，这个返回的ViewHolder类似于之前在getView中的convertView.getTag()，然后在onBindViewHolder方法中去给这个ViewHolder中的控件填充值。其实它的原理跟getView是差不多的，只是做了封装，我们写起来比较简洁。到这里，看到上述运行效果可能有很多人会说，这效果太丑了，连个分隔线都没有，不要急，我们一步一步来。

**3.2 RecyclerView增加分隔线**

      前面我们说到可以通过RecyclerView.addItemDecoration(ItemDecoration decoration)这个方法进行设置，其中它需要的参数就是我们自己定义的继承自ItemDecoration的一个对象。我们可以创建一个继承RecyclerView.ItemDecoration类来绘制分隔线，通过ItemDecoration可以让我们每一个Item从视觉上面相互分开来，例如ListView的divider非常相似的效果。当然像我们上面的例子ItemDecoration我们没有设置也没有报错，那说明ItemDecoration我们并不是强制需要使用，作为我们开发者可以设置或者不设置Decoration的。实现一个ItemDecoration，系统提供的ItemDecoration是一个抽象类，内部除去已经废弃的方法以外，我们主要实现以下三个方法:

**public** **static** **abstract** **class** ItemDecoration {

**public** **void** onDraw(Canvas c,RecyclerView parent,State state) {

          onDraw(c,parent);

      }

**public** **void** onDrawOver(Canvas c,RecyclerView parent,State state) {

          onDrawOver(c,parent);

      }

**public** **void** getItemOffsets(RectoutRect, View view,RecyclerView parent,State state) {

          getItemOffsets(outRect,((LayoutParams)view.getLayoutParams()).getViewLayoutPosition(),parent);

      }

  }

又因为当我们RecyclerView在进行绘制的时候会进行绘制Decoration,那么会去调用onDraw和onDrawOver方法，那么这边我们其实只要去重写onDraw和getItemOffsets这两个方法就可以实现啦。然后LayoutManager会进行Item布局的时候，会去调用getItemOffset方法来计算每个Item的Decoration合适的尺寸，下面我们来具体实现一个Decoration，DividerItemDecoration.**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "http://blog.csdn.net/dmk877/article/details/_blank)**

**package** com.example.reclerviewpractice;

**import** android.content.Context;

**import** android.content.res.TypedArray;

**import** android.graphics.Canvas;

**import** android.graphics.Rect;

**import** android.graphics.drawable.Drawable;

**import** android.support.v7.widget.LinearLayoutManager ;

**import** android.support.v7.widget.RecyclerView;

**import** android.view.View;

**public** **class** DividerItemDecoration **extends** RecyclerView.ItemDecoration {

**private** **static** **final** **int**[] ATTRS = **new** **int**[]{

            android.R.attr. listDivider

    };

**public** **static** **final** **int** HORIZONTAL\_LIST = LinearLayoutManager.HORIZONTAL;

**public** **static** **final** **int** VERTICAL\_LIST = LinearLayoutManager.VERTICAL;

**private** Drawable mDivider;

**private** **int** mOrientation;

**public** DividerItemDecoration(Context context, **int** orientation) {

**final** TypedArray a = context.obtainStyledAttributes(ATTRS );

        mDivider = a.getDrawable(0);

        a.recycle();

        setOrientation(orientation);

    }

**public** **void** setOrientation( **int** orientation) {

**if** (orientation != HORIZONTAL\_LIST && orientation != VERTICAL\_LIST) {

**throw** **new** IllegalArgumentException( "invalid orientation");

        }

        mOrientation = orientation;

    }

    @Override

**public** **void** onDraw(Canvas c, RecyclerView parent) {

**if** (mOrientation == VERTICAL\_LIST) {

            drawVertical(c, parent);

        } **else** {

            drawHorizontal(c, parent);

        }

    }

**public** **void** drawVertical(Canvas c, RecyclerView parent) {

**final** **int** left = parent.getPaddingLeft();

**final** **int** right = parent.getWidth() - parent.getPaddingRight();

**final** **int** childCount = parent.getChildCount();

**for** (**int** i = 0; i < childCount; i++) {

**final** View child = parent.getChildAt(i);

**final** RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child

                    .getLayoutParams();

**final** **int** top = child.getBottom() + params.bottomMargin;

**final** **int** bottom = top + mDivider.getIntrinsicHeight();

            mDivider.setBounds(left, top, right, bottom);

            mDivider.draw(c);

        }

    }

**public** **void** drawHorizontal(Canvas c, RecyclerView parent) {

**final** **int** top = parent.getPaddingTop();

**final** **int** bottom = parent.getHeight() - parent.getPaddingBottom();

**final** **int** childCount = parent.getChildCount();

**for** (**int** i = 0; i < childCount; i++) {

**final** View child = parent.getChildAt(i);

**final** RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child

                    .getLayoutParams();

**final** **int** left = child.getRight() + params.rightMargin;

**final** **int** right = left + mDivider.getIntrinsicHeight();

            mDivider.setBounds(left, top, right, bottom);

            mDivider.draw(c);

        }

    }

    @Override

**public** **void** getItemOffsets(Rect outRect, **int** itemPosition, RecyclerView parent) {

**if** (mOrientation == VERTICAL\_LIST) {

            outRect.set(0, 0, 0, mDivider.getIntrinsicHeight());

        }**else**{

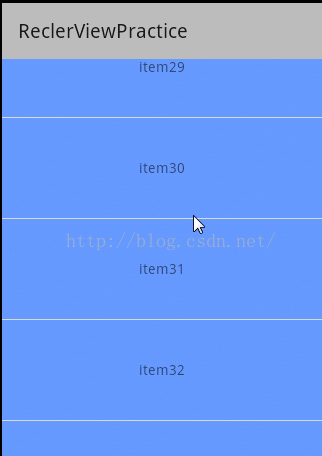
            outRect.set(0, 0, mDivider.getIntrinsicWidth(), 0);

        }

    }

}

在这里我们采用系统主题(android.R.attr.listDivider)来设置成分隔线的，然后来获取尺寸,位置进行setBound(),绘制，接着通过outRect.set()来设置绘制整个区域范围，当然了它是有两种情况的一种LinearLayoutManager.HORIZONTAL另外一种LinearLayoutManager.VERTICAL需要分别对其进行处理，最后不要忘记往RecyclerView中设置该自定义的分割线,然后在MainActivity中加上一句recyclerView .addItemDecoration(new DividerItemDecoration(MainActivity.this,LinearLayoutManager.VERTICAL))即给RecyclerView增加分隔线。然后运行，效果如下：



可以看到已经有了分隔线，跟ListView的效果基本一致了。当然了，既然谷歌给我们提供了这个专门添加分隔线的方法，那它肯定会允许我们自定义分隔线的样式，不然把这个方法抽取出来也没有任何意义。

**3.3 改变分隔线样式**

     那么怎么更改分隔线的样式呢？在上面的DividerItemDecoration这个类中可以看到这个分隔线是跟ListView一样的，即系统的默认的样式，因此我们可以在styles的xml文件中进行更改，更改如下：

<!-- Application theme. -->

<style name ="AppTheme" parent="AppBaseTheme">

    <!-- All customizations that are NOT specific to a particular API-level can go here. -->

    <item name= "android:listDivider">@drawable/divider </item >

</style >

divider的内容如下：

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

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

    android:shape="rectangle" >

        <!-- 填充的颜色 -->

       <solid android:color ="@color/color\_red"/>

      <!--  线条大小 -->

      <size android:height ="1dp" android:width ="1dp"/>

</shape>

修改之后运行效果如下：



可以看到分隔线的颜色变了，当然了这只是一个小例子，我们可以按照业务需求去更改，这样就基本实现了ListView的效果，看到这肯定会有人说，这尼玛，好麻烦，还不如ListView简单呢，从上面的代码量看来确实是使用起来很复杂，但是如果此时你想将这个列表以GridView的形式展示出来，用RecylerView仅仅是换一行代码的事情，

在上面的代码中我们使用了

LinearLayoutManager layoutManager = **new** LinearLayoutManager(**this**);

 //设置布局管理器

recyclerView.setLayoutManager(layoutManager);

RecyclerView.LayoutManager是一个抽象类，系统为我们提供了三个实现类  
①LinearLayoutManager即线性布局，这个是在上面的例子中我们用到的布局  
②GridLayoutManager即表格布局  
③StaggeredGridLayoutManager即流式布局，如瀑布流效果  
假如将上述例子换成GridView的效果，那么相应的代码应该这样改

recyclerView.setLayoutManager(**new** GridLayoutManager(**this**,4));

除此之外上述的分隔线也要做相应的更改，因为在上述DividerItemDecoration这个方法中从

**final** **int** left = parent.getPaddingLeft();

**final** **int** right = parent.getWidth() - parent.getPaddingRight();

这两行我们可以看出来，它是绘制了一条线这条线就是从RecyclerView去掉左右边距后，剩余的部分，因为当显示成ListView时每一行就一个Item所以整体效果看上去就跟ListView差不多，而当展示成GridView那样的效果时，每一行就不止一个条目了，而有可能是多个,所以这个类就不再适用了，我们需要重新写一个，这里我就直接用鸿洋大神写的了，它的博客地址：[http://blog.csdn.net/lmj623565791/article/details/45059587](http://blog.csdn.net/lmj623565791/article/details/45059587" \t "http://blog.csdn.net/dmk877/article/details/_blank)

**package** com.example.reclerviewpractice;

**import** android.content.Context;

**import** android.content.res.TypedArray;

**import** android.graphics.Canvas;

**import** android.graphics.Rect;

**import** android.graphics.drawable.Drawable;

**import** android.support.v7.widget.GridLayoutManager;

**import** android.support.v7.widget.RecyclerView;

**import** android.support.v7.widget.RecyclerView.LayoutManager;

**import** android.support.v7.widget.RecyclerView.State;

**import** android.support.v7.widget.StaggeredGridLayoutManager;

**import** android.view.View;

/\*\*

 \*

 \* @author zhy

 \*

 \*/

**public** **class** DividerGridItemDecoration **extends** RecyclerView.ItemDecoration

{

**private** **static** **final** **int**[] ATTRS = **new** **int**[] { android.R.attr.listDivider };

**private** Drawable mDivider;

**public** DividerGridItemDecoration(Context context)

     {

**final** TypedArray a = context.obtainStyledAttributes(ATTRS );

            mDivider = a.getDrawable(0);

           a.recycle();

     }

     @Override

**public** **void** onDraw(Canvas c, RecyclerView parent, State state)

     {

           drawHorizontal(c, parent);

           drawVertical(c, parent);

     }

**private** **int** getSpanCount(RecyclerView parent)

     {

            // 列数

**int** spanCount = -1;

           LayoutManager layoutManager = parent.getLayoutManager();

**if** (layoutManager **instanceof** GridLayoutManager)

           {

                spanCount = ((GridLayoutManager) layoutManager).getSpanCount();

           } **else** **if** (layoutManager **instanceof** StaggeredGridLayoutManager)

           {

                spanCount = ((StaggeredGridLayoutManager) layoutManager)

                           .getSpanCount();

           }

**return** spanCount;

     }

**public** **void** drawHorizontal(Canvas c, RecyclerView parent)

     {

**int** childCount = parent.getChildCount();

**for** ( **int** i = 0; i < childCount; i++)

           {

**final** View child = parent.getChildAt(i);

**final** RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child

                           .getLayoutParams();

**final** **int** left = child.getLeft() - params.leftMargin;

**final** **int** right = child.getRight() + params.rightMargin

                           + mDivider.getIntrinsicWidth();

**final** **int** top = child.getBottom() + params.bottomMargin;

**final** **int** bottom = top + mDivider.getIntrinsicHeight();

                 mDivider.setBounds(left, top, right, bottom);

                 mDivider.draw(c);

           }

     }

**public** **void** drawVertical(Canvas c, RecyclerView parent)

     {

**final** **int** childCount = parent.getChildCount();

**for** ( **int** i = 0; i < childCount; i++)

           {

**final** View child = parent.getChildAt(i);

**final** RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child

                           .getLayoutParams();

**final** **int** top = child.getTop() - params.topMargin;

**final** **int** bottom = child.getBottom() + params.bottomMargin;

**final** **int** left = child.getRight() + params.rightMargin;

**final** **int** right = left + mDivider.getIntrinsicWidth();

                 mDivider.setBounds(left, top, right, bottom);

                 mDivider.draw(c);

           }

     }

**private** **boolean** isLastColum(RecyclerView parent, **int** pos, **int** spanCount,

**int** childCount)

     {

           LayoutManager layoutManager = parent.getLayoutManager();

**if** (layoutManager **instanceof** GridLayoutManager)

           {

**if** ((pos + 1) % spanCount == 0) // 如果是最后一列，则不需要绘制右边

                {

**return** **true**;

                }

           } **else** **if** (layoutManager **instanceof** StaggeredGridLayoutManager)

           {

**int** orientation = ((StaggeredGridLayoutManager) layoutManager)

                           .getOrientation();

**if** (orientation == StaggeredGridLayoutManager.VERTICAL )

                {

**if** ((pos + 1) % spanCount == 0) // 如果是最后一列，则不需要绘制右边

                     {

**return** **true**;

                     }

                } **else**

                {

                     childCount = childCount - childCount % spanCount;

**if** (pos >= childCount) // 如果是最后一列，则不需要绘制右边

**return** **true**;

                }

           }

**return** **false**;

     }

**private** **boolean** isLastRaw(RecyclerView parent, **int** pos, **int** spanCount,

**int** childCount)

     {

           LayoutManager layoutManager = parent.getLayoutManager();

**if** (layoutManager **instanceof** GridLayoutManager)

           {

                childCount = childCount - childCount % spanCount;

**if** (pos >= childCount) // 如果是最后一行，则不需要绘制底部

**return** **true**;

           } **else** **if** (layoutManager **instanceof** StaggeredGridLayoutManager)

           {

**int** orientation = ((StaggeredGridLayoutManager) layoutManager)

                           .getOrientation();

                 // StaggeredGridLayoutManager 且纵向滚动

**if** (orientation == StaggeredGridLayoutManager.VERTICAL )

                {

                     childCount = childCount - childCount % spanCount;

                      // 如果是最后一行，则不需要绘制底部

**if** (pos >= childCount)

**return** **true**;

                } **else**

                 // StaggeredGridLayoutManager 且横向滚动

                {

                      // 如果是最后一行，则不需要绘制底部

**if** ((pos + 1) % spanCount == 0)

                     {

**return** **true**;

                     }

                }

           }

**return** **false**;

     }

     @Override

**public** **void** getItemOffsets(Rect outRect, **int** itemPosition,

                RecyclerView parent)

     {

**int** spanCount = getSpanCount(parent);

**int** childCount = parent.getAdapter().getItemCount();

**if** (isLastRaw(parent, itemPosition, spanCount, childCount))// 如果是最后一行，则不需要绘制底部

           {

                outRect.set(0, 0, mDivider.getIntrinsicWidth(), 0);

           } **else** **if** (isLastColum(parent, itemPosition, spanCount, childCount))// 如果是最后一列，则不需要绘制右边

           {

                outRect.set(0, 0, 0, mDivider.getIntrinsicHeight());

           } **else**

           {

                outRect.set(0, 0, mDivider.getIntrinsicWidth(),

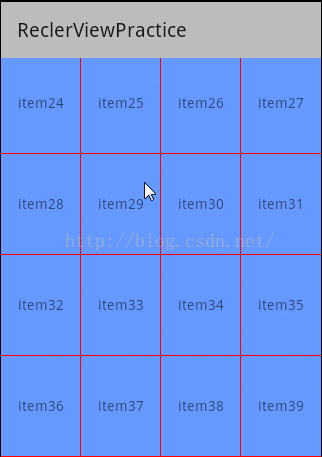
                            mDivider.getIntrinsicHeight());

           }

     }

}

别忘了更改分隔线recyclerView .addItemDecoration(**new** DividerGridItemDecoration(**this** ));之后运行发现效果如下



可以看到如果你准备好了分隔线的这个类，从ListView效果到GridView效果，只需要几行代码，是不是瞬间感觉高大上了？还有更让人瞠目结舌的效果，将上述代码做如下更改

StaggeredGridLayoutManager layoutManager = **new** StaggeredGridLayoutManager(4,StaggeredGridLayoutManager.HORIZONTAL);

//设置布局管理器

recyclerView.setLayoutManager(layoutManager);

这里需要注意的是StaggeredGridLayoutManager构造的第二个参数传一个orientation，如果传入的是StaggeredGridLayoutManager.VERTICAL那么前面那个参数就代表有多少列；如果传是StaggeredGridLayoutManager.HORIZONTAL那么前面那个参数就代表有多少行

运行效果如下



这效果是不是有点逆天？可以看到，固定为4行，变成了左右滑动。有一点需要注意，如果是横向的时候，item的宽度需要注意去设置，毕竟横向的宽度没有约束了，因为控件可以横向滚动了，另外它还可以实现瀑布流的效果，关于瀑布流我准备后面专门写一篇博客。  
**3.4 RecyclerView增加和删除的动画(包括RecyclerView.Adapter中刷新的几个方法的对比)**  
在上面也提到了控制RecyclerView增加和删除的动画是通过ItemAnimator这个类来实现的，ItemAnimator这类也是个抽象的类，系统默认给我们提供了一种增加和删除的动画，下面我们就来看看这种动画的效果，我们需要做的修改如下：

LinearLayoutManager layoutManager = **new** LinearLayoutManager(**this**);

 //设置布局管理器

recyclerView.setLayoutManager(layoutManager);

//设置增加或删除条目的动画

recyclerView.setItemAnimator( **new** DefaultItemAnimator());

然后重写ActionBar的

@Override

**public** **boolean** onCreateOptionsMenu(Menu menu)

{

     getMenuInflater().inflate(R.menu. main, menu);

**return** **super**.onCreateOptionsMenu(menu);

}

@Override

**public** **boolean** onOptionsItemSelected(MenuItem item)

{

**switch** (item.getItemId())

     {

**case** R.id. id\_action\_add:

               recycleAdapter.addData(1);

**break**;

**case** R.id. id\_action\_delete:

               recycleAdapter.removeData(1);

**break**;

     }

**return** **true**;

}

关于R.menu.main中的main.xml这个文件代码就不贴了，在最后的一个汇总的例子里会有

recyclerViewAdapter中增加的两个方法：

**public** **void** addData( **int** position) {

          mDatas.add(position, "Insert One");

          notifyItemInserted(position);

          notifyItemRangeChanged(position, mDatas.size());

     }

**public** **void** removeData( **int** position) {

          mDatas.remove(position);

          notifyItemRemoved(position);  
          notifyItemRangeChanged(position, mDatas.size());

     }

这里需要说一下RecyclerView.Adapter中刷新数据的几个方法，一共有这么几个方法

**public** **final** **void** notifyDataSetChanged()

**public** **final** **void** notifyItemChanged(**int** position)

**public** **final** **void** notifyItemRangeChanged(**int** positionStart, **int** itemCount)

**public** **final** **void** notifyItemInserted(**int** position)

**public** **final** **void** notifyItemMoved(**int** fromPosition, **int** toPosition)

**public** **final** **void** notifyItemRangeInserted(**int** positionStart, **int** itemCount)

**public** **final** **void** notifyItemRemoved(**int** position)

**public** **final** **void** notifyItemRangeRemoved(**int** positionStart, **int** itemCount)

**notifyDataSetChanged()**这个方法跟我们平时用到的ListView的Adapter的方法一样，这里就不多做描述了。

**notifyItemChanged(int position)**，当position位置的数据发生了改变时就会调用这个方法，就会回调对应position的onBindViewHolder()方法了，当然，因为ViewHolder是复用的，所以如果position在当前屏幕以外，也就不会回调了，因为没有意义，下次position滚动会当前屏幕以内的时候同样会调用onBindViewHolder()方法刷新数据了。其他的方法也是同样的道理。

public final void notifyItemRangeChanged(int positionStart, int itemCount)，顾名思义，可以刷新从positionStart开始itemCount数量的item了（这里的刷新指回调onBindViewHolder()方法）。

public final void notifyItemInserted(int position)，这个方法是在第position位置被插入了一条数据的时候可以使用这个方法刷新，注意这个方法调用后会有插入的动画，这个动画可以使用默认的，也可以自己定义。

public final void notifyItemMoved(int fromPosition, int toPosition)，这个方法是从fromPosition移动到toPosition为止的时候可以使用这个方法刷新

public final void notifyItemRangeInserted(int positionStart, int itemCount)，显然是批量添加。

public final void notifyItemRemoved(int position)，第position个被删除的时候刷新，同样会有动画。  
将上述更改运行，点击添加和删除按钮效果图如下：

public final void notifyItemRangeRemoved(int positionStart, int itemCount)，批量删除。

可以看到系统给提供的动画效果还不错，当然我们也可以按照业务需求去自己定义动画效果。

**3.5 给RecyclerView的Item添加点击事件**

    到这里还有一点从文章开头到现在我们都没有提及，就是Item的点击事件RecyclerView监听事件处理在ListView使用的时候，该控件给我们提供一个onItemClickListener监听器，这样当我们点击Item的时候，会回调相关的方法，以便我们方便处理Item点击事件。对于RecyclerView来讲，非常可惜的是，该控件没有给我们提供这样的内置监听器方法，不过我们可以进行改造实现，可以这样实现Item的点击事件的监听，在我们的adapter中增加这两个方法

**public** **interface** OnItemClickListener{

**void** onClick( **int** position);

**void** onLongClick( **int** position);

     }

**public** **void** setOnItemClickListener(OnItemClickListener onItemClickListener ){

**this**. mOnItemClickListener=onItemClickListener;

     }

然后onBindViewHolder方法要做如下更改

@Override

**public** **void** onBindViewHolder(MyViewHolder holder, **final** **int** position) {

     holder. tv.setText( mDatas.get(position));

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

          holder. itemView.setOnClickListener( **new** OnClickListener() {

               @Override

**public** **void** onClick(View v) {

                    mOnItemClickListener.onClick(position);

               }

          });

          holder.itemView.setOnLongClickListener( **new** OnLongClickListener() {

               @Override

**public** **boolean** onLongClick(View v) {

                    mOnItemClickListener.onLongClick(position);

**return** **false**;

               }

          });

     }

}

在MainAcitivity中增加

recycleAdapter.setOnItemClickListener(**new** OnItemClickListener() {

    @Override

**public** **void** onLongClick(**int** position) {

         Toast.makeText(MainActivity.**this**,"onLongClick事件       您点击了第："+position+"个Item",0).show();

    }

    @Override

**public** **void** onClick(**int** position) {

         Toast.makeText(MainActivity.**this**,"onClick事件       您点击了第："+position+"个Item",0).show();

    }

});

然后运行，效果如下：

