Android 使用开源库StickyGridHeaders来实现带sections和headers的GridView显示本地图片效果

标签： StickyGridHeaders headers sections OOM 显示图片

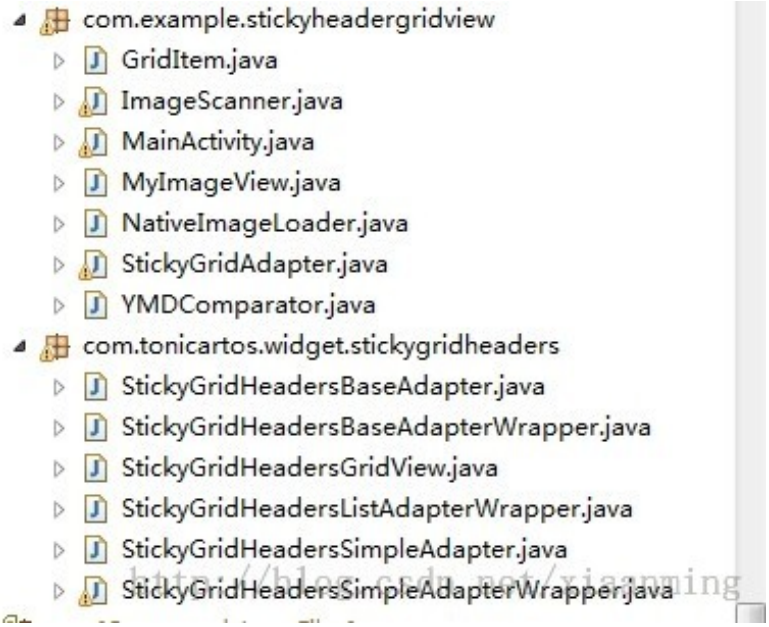
2014-03-06 09:04 21429人阅读 评论(41) 收藏 举报

分类： Android 高手进阶（21）

转载请注明本文出自xiaanming的博客（http://blog.csdn.net/xiaanming/article/details/20481185），请尊重他人的辛勤劳动成果，谢谢！

大家好！过完年回来到现在差不多一个月没写文章了，一是觉得不知道写哪些方面的文章，没有好的题材来写，二是因为自己的一些私事给耽误了，所以过完年的第一篇文章到现在才发表出来，2014年我还是会继续在CSDN上面更新我的博客，欢迎大家关注一下，今天这篇文章主要的是介绍下开源库StickyGridHeaders的使用，StickyGridHeaders是一个自定义GridView带sections和headers的Android库，sections就是GridView item之间的分隔，headers就是固定在GridView顶部的标题，类似一些Android手机联系人的效果，StickyGridHeaders的介绍在https://github.com/TonicArtos/StickyGridHeaders，与此对应也有一个相同效果的自定义ListView带sections和headers的开源库https://github.com/emilsjolander/StickyListHeaders，大家有兴趣的可以去看下，我这里介绍的是StickyGridHeaders的使用，我在Android应用方面看到使用StickyGridHeaders的不是很多，而是在Iphone上看到相册采用的是这种效果，于是我就使用StickyGridHeaders来仿照Iphone按照日期分隔显示本地图片

我们先新建一个Android项目StickyHeaderGridView，去https://github.com/TonicArtos/StickyGridHeaders下载开源库，为了方便浏览源码我直接将源码拷到我的工程中了



com.tonicartos.widget.stickygridheaders这个包就是我放StickyGridHeaders开源库的源码，com.example.stickyheadergridview这个包是我实现此功能的代码，类看起来还蛮多的，下面我就一一来介绍了GridItem用来封装StickyGridHeadersGridView 每个Item的数据，里面有本地图片的路径，图片加入手机系统的时间和headerId

```
[java]
```

```java
package com.example.stickyheadergridview;
/**
 * @blog http://blog.csdn.net/xiaanming
 *
 * @author xiaanming
 *
 */
public class GridItem {
    /**
     * 图片的路径
     */
    private String path;
    /**
     * 图片加入手机中的时间，只取了年月日
     */
    private String time;
    /**
     * 每个Item对应的HeaderId
     */
    private int headerId;

    public GridItem(String path, String time) {
        super();
        this.path = path;
        this.time = time;
    }

    public String getPath() {
        return path;
    }
    public void setPath(String path) {
        this.path = path;
    }
    public String getTime() {
        return time;
    }
    public void setTime(String time) {
        this.time = time;
    }

    public int getHeaderId() {
        return headerId;
    }

    public void setHeaderId(int headerId) {
        this.headerId = headerId;
    }


}
```

图片的路径path和图片加入的时间time 我们直接可以通过ContentProvider获取，但是headerId需要我们根据逻辑来生成。

```
[java]
```

```
01. package com.example.stickyheadergridview;
02. 
03. import android.content.ContentResolver;
04. import android.content.Context;
05. import android.content.Intent;
06. import android.database.Cursor;
07. import android.net.Uri;
08. import android.os.Environment;
09. import android.os.Handler;
10. import android.os.Message;
11. import android.provider.MediaStore;
12. /**
13.  * 图片扫描器
14.  *
15.  * @author xiaanming
16.  *
17.  */
18. public class ImageScanner {
19.     private Context mContext;
20. 
21.     public ImageScanner(Context context){
22.         this.mContext = context;
23.     }
24. 
25.     /**
26.      * 利用ContentProvider扫描手机中的图片，将扫描的Cursor回调到ScanCompleteCallBack
27.      * 接口的scanComplete方法中，此方法在运行在子线程中
28.      */
29.     public void scanImages(final ScanCompleteCallBack callback) {
30.         final Handler mHandler = new Handler() {
31. 
32.             @Override
33.             public void handleMessage(Message msg) {
34.                 super.handleMessage(msg);
35.                 callback.scanComplete((Cursor)msg.obj);
36.             }
37.         };
38. 
39.         new Thread(new Runnable() {
40. 
41.             @Override
42.             public void run() {
43.                 //先发送广播扫描下整个sd卡
44.                 mContext.sendBroadcast(new Intent(
45.                             Intent.ACTION_MEDIA_MOUNTED,
46.                             Uri.parse("file://" + Environment.getExternalStorageDirectory(
47. 
48.                 Uri mImageUri = MediaStore.Images.Media.EXTERNAL_CONTENT_URI;
49.                 ContentResolver mContentResolver = mContext.getContentResolver();
50. 
51.                 Cursor mCursor = mContentResolver.query(mImageUri, null, null, null, Media
52. 
53.                 //利用Handler通知调用线程
54.                 Message msg = mHandler.obtainMessage();
55.                 msg.obj = mCursor;
56.                 mHandler.sendMessage(msg);
57.             }
```

```
        }).start();

    }

    /**
     * 扫描完成之后的回调接口
     *
     */
    public static interface ScanCompleteCallBack{
        public void scanComplete(Cursor cursor);
    }


}
```

ImageScanner是一个图片的扫描器类，该类使用ContentProvider扫描手机中的图片，我们通过调用scanImages()方法就能对手机中的图片进行扫描，将扫描的Cursor回调到ScanCompleteCallBack 接口的scanComplete方法中，由于考虑到扫描图片属于耗时操作，所以该操作运行在子线程中，在我们扫描图片之前我们需要先发送广播来扫描外部媒体库，为什么要这么做呢，假如我们新增加一张图片到sd卡，图片确实已经添加了进去，但是我们此时的媒体库还没有同步更新，若不同步媒体库我们就看不到新增加的图片，当然我们可以通过重新启动系统来更新媒体库，但是这样不可取，所以我们直接发送广播就可以同步媒体库了。

[java]

```java
package com.example.stickyheadergridview;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Point;
import android.os.Handler;
import android.os.Message;
import android.support.v4.util.LruCache;
import android.util.Log;

/**
 * 本地图片加载器,采用的是异步解析本地图片，单例模式利用getInstance()获取NativeImageLoader实例
 * 调用loadNativeImage()方法加载本地图片，此类可作为一个加载本地图片的工具类
 *
 * @blog http://blog.csdn.net/xiaanming
 *
 * @author xiaanming
 *
 */
public class NativeImageLoader {
    private static final String TAG = NativeImageLoader.class.getSimpleName();
    private static NativeImageLoader mInstance = new NativeImageLoader();
    private static LruCache<String, Bitmap> mMemoryCache;
    private ExecutorService mImageThreadPool = Executors.newFixedThreadPool(1);


    private NativeImageLoader(){
```

```
        //获取应用程序的最大内存
        final int maxMemory = (int) (Runtime.getRuntime().maxMemory());

        //用最大内存的1/8来存储图片
        final int cacheSize = maxMemory / 8;
        mMemoryCache = new LruCache<String, Bitmap>(cacheSize) {

            //获取每张图片的bytes
            @Override
            protected int sizeOf(String key, Bitmap bitmap) {
                return bitmap.getRowBytes() * bitmap.getHeight();
            }

        };
    }

    /**
     * 通过此方法来获取NativeImageLoader的实例
     * @return
     */
    public static NativeImageLoader getInstance(){
        return mInstance;
    }


    /**
     * 加载本地图片，对图片不进行裁剪
     * @param path
     * @param mCallBack
     * @return
     */
    public Bitmap loadNativeImage(final String path, final NativeImageCallBack mCallBack)
{
        return this.loadNativeImage(path, null, mCallBack);
    }

    /**
     * 此方法来加载本地图片，这里的mPoint是用来封装ImageView的宽和高，我们会根据ImageView控件的
大小来裁剪Bitmap
     * 如果你不想裁剪图片，调用
loadNativeImage(final String path, final NativeImageCallBack mCallBack)来加载
     * @param path
     * @param mPoint
     * @param mCallBack
     * @return
     */
    public Bitmap loadNativeImage(final String path, final Point mPoint, final NativeImage
{
        //先获取内存中的Bitmap
        Bitmap bitmap = getBitmapFromMemCache(path);

        final Handler mHander = new Handler(){

            @Override
            public void handleMessage(Message msg) {
                super.handleMessage(msg);
                mCallBack.onImageLoader((Bitmap)msg.obj, path);
```

```
            }

        };

        //若该Bitmap不在内存缓存中，则启用线程去加载本地的图片，并将Bitmap加入到mMemoryCache
中
        if(bitmap == null){
            mImageThreadPool.execute(new Runnable() {

                @Override
                public void run() {
                    //先获取图片的缩略图
                    Bitmap mBitmap = decodeThumbBitmapForFile(path, mPoint == null ? 0: mP
                    Message msg = mHander.obtainMessage();
                    msg.obj = mBitmap;
                    mHander.sendMessage(msg);

                    //将图片加入到内存缓存
                    addBitmapToMemoryCache(path, mBitmap);
                }
            });
        }
        return bitmap;

    }



    /**
     * 往内存缓存中添加Bitmap
     *
     * @param key
     * @param bitmap
     */
    private void addBitmapToMemoryCache(String key, Bitmap bitmap) {
        if (getBitmapFromMemCache(key) == null && bitmap != null) {
            mMemoryCache.put(key, bitmap);
        }
    }

    /**
     * 根据key来获取内存中的图片
     * @param key
     * @return
     */
    private Bitmap getBitmapFromMemCache(String key) {
        Bitmap bitmap = mMemoryCache.get(key);

        if(bitmap != null){
            Log.i(TAG, "get image for LRUCache , path = " + key);
        }
        return bitmap;
    }

    /**
     * 清除LruCache中的bitmap
     */
```

```
    public void trimMemCache(){
        mMemoryCache.evictAll();
    }


    /**
     * 根据View(主要是ImageView)的宽和高来获取图片的缩略图
     * @param path
     * @param viewWidth
     * @param viewHeight
     * @return
     */
    private Bitmap decodeThumbBitmapForFile(String path, int viewWidth, int viewHeight)
{
        BitmapFactory.Options options = new BitmapFactory.Options();
        //设置为true,表示解析Bitmap对象，该对象不占内存
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFile(path, options);
        //设置缩放比例
        options.inSampleSize = computeScale(options, viewWidth, viewHeight);

        //设置为false,解析Bitmap对象加入到内存中
        options.inJustDecodeBounds = false;


        Log.e(TAG, "get Iamge form file,  path = " + path);

        return BitmapFactory.decodeFile(path, options);
    }


    /**
     * 根据View(主要是ImageView)的宽和高来计算Bitmap缩放比例。默认不缩放
     * @param options
     * @param width
     * @param height
     */
    private int computeScale(BitmapFactory.Options options, int viewWidth, int viewHeight)
{
        int inSampleSize = 1;
        if(viewWidth == 0 || viewWidth == 0){
            return inSampleSize;
        }
        int bitmapWidth = options.outWidth;
        int bitmapHeight = options.outHeight;

        //假如Bitmap的宽度或高度大于我们设定图片的View的宽高，则计算缩放比例
        if(bitmapWidth > viewWidth || bitmapHeight > viewWidth){
            int widthScale = Math.round((float) bitmapWidth / (float) viewWidth);
            int heightScale = Math.round((float) bitmapHeight / (float) viewWidth);

            //为了保证图片不缩放变形，我们取宽高比例最小的那个
            inSampleSize = widthScale < heightScale ? widthScale : heightScale;
        }
        return inSampleSize;
    }

```

```
194.
195.     /**
196.      * 加载本地图片的回调接口
197.      *
198.      * @author xiaanming
199.      *
200.      */
201.     public interface NativeImageCallBack{
202.         /**
203.          * 当子线程加载完了本地的图片，将Bitmap和图片路径回调在此方法中
204.          * @param bitmap
205.          * @param path
206.          */
207.         public void onImageLoader(Bitmap bitmap, String path);
208.     }
209. }
```

NativeImageLoader该类是一个单例类，提供了本地图片加载，内存缓存，裁剪等逻辑，该类在加载本地图片的时候采用的是异步加载的方式，对于大图片的加载也是比较耗时的，所以采用子线程的方式去加载，对于图片的缓存机制使用的是LruCache，我们使用手机分配给应用程序内存的1/8用来缓存图片，给图片缓存的内存不宜太大，太大也可能会发生OOM，该类是用我之前写的文章Android 使用ContentProvider扫描手机中的图片，仿微信显示本地图片效果，在这里我就不做过多的介绍，有兴趣的可以去看看那篇文章，不过这里新增了一个方法trimMemCache(),，用来清空LruCache使用的内存

我们看主界面的布局代码，里面只有一个自定义的StickyGridHeadersGridView控件

```
[html]
01. <?xml version="1.0" encoding="utf-8"?>
02. <com.tonicartos.widget.stickygridheaders.StickyGridHeadersGridView xmlns:android="http://s
03.     xmlns:tools="http://schemas.android.com/tools"
04.     android:id="@+id/asset_grid"
05.     android:layout_width="match_parent"
06.     android:layout_height="match_parent"
07.     android:clipToPadding="false"
08.     android:columnWidth="90dip"
09.     android:horizontalSpacing="3dip"
10.     android:numColumns="auto_fit"
11.     android:verticalSpacing="3dip" />
```

在看主界面的代码之前我们先看StickyGridAdapter的代码

```
[java]
01. package com.example.stickyheadergridview;
02.
03. import java.util.List;
04.
05. import android.content.Context;
06. import android.graphics.Bitmap;
07. import android.graphics.Point;
08. import android.view.LayoutInflater;
09. import android.view.View;
```

```
10. import android.view.ViewGroup;
11. import android.widget.BaseAdapter;
12. import android.widget.GridView;
13. import android.widget.ImageView;
14. import android.widget.TextView;
15. 
16. import com.example.stickyheadergridview.MyImageView.OnMeasureListener;
17. import com.example.stickyheadergridview.NativeImageLoader.NativeImageCallBack;
18. import com.tonicartos.widget.stickygridheaders.StickyGridHeadersSimpleAdapter;
19. /**
20.  * StickyHeaderGridView的适配器，除了要继承BaseAdapter之外还需要
21.  * 实现StickyGridHeadersSimpleAdapter接口
22.  *
23.  * @blog http://blog.csdn.net/xiaanming
24.  *
25.  * @author xiaanming
26.  *
27.  */
28. public class StickyGridAdapter extends BaseAdapter implements
29.         StickyGridHeadersSimpleAdapter {
30. 
31.     private List<GridItem> hasHeaderIdList;
32.     private LayoutInflater mInflater;
33.     private GridView mGridView;
34.     private Point mPoint = new Point(0, 0);//用来封装ImageView的宽和高的对象
35. 
36.     public StickyGridAdapter(Context context, List<GridItem> hasHeaderIdList,
37.             GridView mGridView) {
38.         mInflater = LayoutInflater.from(context);
39.         this.mGridView = mGridView;
40.         this.hasHeaderIdList = hasHeaderIdList;
41.     }
42. 
43. 
44.     @Override
45.     public int getCount() {
46.         return hasHeaderIdList.size();
47.     }
48. 
49.     @Override
50.     public Object getItem(int position) {
51.         return hasHeaderIdList.get(position);
52.     }
53. 
54.     @Override
55.     public long getItemId(int position) {
56.         return position;
57.     }
58. 
59.     @Override
60.     public View getView(int position, View convertView, ViewGroup parent) {
61.         ViewHolder mViewHolder;
62.         if (convertView == null) {
63.             mViewHolder = new ViewHolder();
64.             convertView = mInflater.inflate(R.layout.grid_item, parent, false);
65.             mViewHolder.mImageView = (MyImageView) convertView
66.                     .findViewById(R.id.grid_item);
```

```
            convertView.setTag(mViewHolder);

             //用来监听ImageView的宽和高
            mViewHolder.mImageView.setOnMeasureListener(new OnMeasureListener() {

                @Override
                public void onMeasureSize(int width, int height) {
                    mPoint.set(width, height);
                }
            });

        } else {
            mViewHolder = (ViewHolder) convertView.getTag();
        }

        String path = hasHeaderIdList.get(position).getPath();
        mViewHolder.mImageView.setTag(path);

        Bitmap bitmap = NativeImageLoader.getInstance().loadNativeImage(path, mPoint,
                new NativeImageCallBack() {

                    @Override
                    public void onImageLoader(Bitmap bitmap, String path) {
                        ImageView mImageView = (ImageView) mGridView
                                .findViewWithTag(path);
                        if (bitmap != null && mImageView != null) {
                            mImageView.setImageBitmap(bitmap);
                        }
                    }
                });

        if (bitmap != null) {
            mViewHolder.mImageView.setImageBitmap(bitmap);
        } else {
            mViewHolder.mImageView.setImageResource(R.drawable.friends_sends_pictures_no);
        }

        return convertView;
    }


    @Override
    public View getHeaderView(int position, View convertView, ViewGroup parent) {
        HeaderViewHolder mHeaderHolder;

        if (convertView == null) {
            mHeaderHolder = new HeaderViewHolder();
            convertView = mInflater.inflate(R.layout.header, parent, false);
            mHeaderHolder.mTextView = (TextView) convertView
                    .findViewById(R.id.header);
            convertView.setTag(mHeaderHolder);
        } else {
            mHeaderHolder = (HeaderViewHolder) convertView.getTag();
        }
        mHeaderHolder.mTextView.setText(hasHeaderIdList.get(position).getTime());

        return convertView;
```

```
    }

    /**
     * 获取HeaderId, 只要HeaderId不相等就添加一个Header
     */
    @Override
    public long getHeaderId(int position) {
        return hasHeaderIdList.get(position).getHeaderId();
    }


    public static class ViewHolder {
        public MyImageView mImageView;
    }

    public static class HeaderViewHolder {
        public TextView mTextView;
    }


}
```

除了要继承BaseAdapter之外还需要实现StickyGridHeadersSimpleAdapter接口，继承BaseAdapter需要实现getCount()，getItem(int position)， getItemId(int position)，getView(int position, View convertView, ViewGroup parent)这四个方法，这几个方法的实现跟我们平常实现的方式一样，主要是看一下getView()方法，我们将每个item的图片路径设置Tag到该ImageView上面，然后利用NativeImageLoader来加载本地图片，在这里使用的ImageView依然是自定义的MyImageView，该自定义ImageView主要实现当MyImageView测量完毕之后，就会将测量的宽和高回调到onMeasureSize()中，然后我们可以根据MyImageView的大小来裁剪图片
另外我们需要实现StickyGridHeadersSimpleAdapter接口的getHeaderId(int position)和getHeaderView(int position, View convertView, ViewGroup parent)，getHeaderId(int position)方法返回每个Item的headerId，getHeaderView()方法是生成sections和headers的，如果某个item的headerId跟他下一个item的HeaderId不同，则会调用getHeaderView方法生成一个sections用来区分不同的组，还会根据firstVisibleItem的headerId来生成一个位于顶部的headers，所以如何生成每个Item的headerId才是关键，生成headerId的方法在MainActivity中

```
[java]
package com.example.stickyheadergridview;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.ListIterator;
import java.util.Map;
import java.util.TimeZone;

import android.app.Activity;
import android.app.ProgressDialog;
```

```
15. import android.database.Cursor;
16. import android.os.Bundle;
17. import android.provider.MediaStore;
18. import android.widget.GridView;
19. 
20. import com.example.stickyheadergridview.ImageScanner.ScanCompleteCallBack;
21. 
22. public class MainActivity extends Activity {
23.     private ProgressDialog mProgressDialog;
24.     /**
25.      * 图片扫描器
26.      */
27.     private ImageScanner mScanner;
28.     private GridView mGridView;
29.     /**
30.      * 没有HeaderId的List
31.      */
32.     private List<GridItem> nonHeaderIdList = new ArrayList<GridItem>();
33. 
34. 
35.     @Override
36.     protected void onCreate(Bundle savedInstanceState) {
37.         super.onCreate(savedInstanceState);
38.         setContentView(R.layout.activity_main);
39. 
40.         mGridView = (GridView) findViewById(R.id.asset_grid);
41.         mScanner = new ImageScanner(this);
42. 
43.         mScanner.scanImages(new ScanCompleteCallBack() {
44.             {
45.                 mProgressDialog = ProgressDialog.show(MainActivity.this, null, "正在加
    载...");
46.             }
47. 
48.             @Override
49.             public void scanComplete(Cursor cursor) {
50.                 // 关闭进度条
51.                 mProgressDialog.dismiss();
52. 
53.                 if(cursor == null){
54.                     return;
55.                 }
56. 
57.                 while (cursor.moveToNext()) {
58.                     // 获取图片的路径
59.                     String path = cursor.getString(cursor
60.                             .getColumnIndex(MediaStore.Images.Media.DATA));
61.                     //获取图片的添加到系统的毫秒数
62.                     long times = cursor.getLong(cursor
63.                             .getColumnIndex(MediaStore.Images.Media.DATE_ADDED));
64. 
65.                     GridItem mGridItem = new GridItem(path, paserTimeToYMD(times, "yyyy年
    MM月dd日"));
66.                     nonHeaderIdList.add(mGridItem);
67. 
68.                 }
69.                 cursor.close();
```

```
70.
71.                 //给GridView的item的数据生成HeaderId
72.                 List<GridItem> hasHeaderIdList = generateHeaderId(nonHeaderIdList);
73.                 //排序
74.                 Collections.sort(hasHeaderIdList, new YMDComparator());
75.                 mGridView.setAdapter(new StickyGridAdapter(MainActivity.this, hasHeaderIdL
76.
77.             }
78.         });
79.     }
80.
81.
82.     /**
83.      * 对GridView的Item生成HeaderId, 根据图片的添加时间的年、月、日来生成HeaderId
84.      * 年、月、日相等HeaderId就相同
85.      * @param nonHeaderIdList
86.      * @return
87.      */
88.     private List<GridItem> generateHeaderId(List<GridItem> nonHeaderIdList) {
89.         Map<String, Integer> mHeaderIdMap = new HashMap<String, Integer>();
90.         int mHeaderId = 1;
91.         List<GridItem> hasHeaderIdList;
92.
93.         for(ListIterator<GridItem> it = nonHeaderIdList.listIterator(); it.hasNext();){
94.             GridItem mGridItem = it.next();
95.             String ymd = mGridItem.getTime();
96.             if(!mHeaderIdMap.containsKey(ymd)){
97.                 mGridItem.setHeaderId(mHeaderId);
98.                 mHeaderIdMap.put(ymd, mHeaderId);
99.                 mHeaderId ++;
100.             }else{
101.                 mGridItem.setHeaderId(mHeaderIdMap.get(ymd));
102.             }
103.         }
104.         hasHeaderIdList = nonHeaderIdList;
105.
106.         return hasHeaderIdList;
107.     }
108.
109.
110.     @Override
111.     protected void onDestroy() {
112.         super.onDestroy();
113.         //退出页面清除LRUCache中的Bitmap占用的内存
114.         NativeImageLoader.getInstance().trimMemCache();
115.     }
116.
117.
118.     /**
119.      * 将毫秒数装换成pattern这个格式，我这里是转换成年月日
120.      * @param time
121.      * @param pattern
122.      * @return
123.      */
124.     public static String paserTimeToYMD(long time, String pattern ) {
125.         System.setProperty("user.timezone", "Asia/Shanghai");
126.         TimeZone tz = TimeZone.getTimeZone("Asia/Shanghai");
```

```
127.          TimeZone.setDefault(tz);
128.          SimpleDateFormat format = new SimpleDateFormat(pattern);
129.          return format.format(new Date(time * 1000L));
130.      }
131.
132.  }
```

主界面的代码主要是组装StickyGridHeadersGridView的数据，我们将扫描出来的图片的路径，时间的毫秒数解析成年月日的格式封装到GridItem中，然后将GridItem加入到List中，此时每个Item还没有生成headerId，我们需要调用generateHeaderId()，该方法主要是将同一天加入的系统的图片生成相同的HeaderId,这样子同一天加入的图片就在一个组中，当然你要改成同一个月的图片在一起，修改paserTimeToYMD（）方法的第二个参数就行了，当Activity finish之后，我们利用NativeImageLoader.getInstance().trimMemCache()释放内存，当然我们还需要对GridView的数据进行排序，比如说headerId相同的item不连续，headerId相同的item就会生成多个sections(即多个分组)，所以我们要利用YMDComparator使得在同一天加入的图片在一起，YMDComparator的代码如下

```java
package com.example.stickyheadergridview;

import java.util.Comparator;

public class YMDComparator implements Comparator<GridItem> {

    @Override
    public int compare(GridItem o1, GridItem o2) {
        return o1.getTime().compareTo(o2.getTime());
    }

}
```

当然这篇文章不使用YMDComparator也是可以的，因为我在利用ContentProvider获取图片的时候，就是根据加入系统的时间排序的，排序只是针对一般的数据来说的。

接下来我们运行下程序看看效果如何

今天的文章就到这里结束了，感谢大家的观看，上面还有一个类和一些资源文件没有贴出来，大家有兴趣研究下就直接下载项目源码，记住采用LruCache缓存图片的时候，cacheSize不要设置得过大，不然产生OOM的概率就更大些，我利用上面的程序测试显示600多张图片来回滑动，没有产生OOM,有问题不明白的同学可以在下面留言！

项目源码，点击下载