

[java]

[java]

```
01. <span style="font-size:14px;">    if (convertView == null) {
02.         convertView = LayoutInflater.from(context).inflate(
03.             R.layout.personplans_item, parent, false);
04.     }
05.
06.
07.     TextView tv_product_type1 = BaseViewHolder.get(convertView,
08.         R.id.tv_product_type1); </span>
```

注意：在BaseViewHolder类中我们看到SparseArray是Android提供的一个工具类，用意是用来取代HashMap工具类的。如下图：

```
DistrictCollectAdapter(List<DistrictInfo> distrs,Context ctx){
    this.distrs = distrs;
    this.inflater = LayoutInflater.from(ctx);
    Use new SparseArray<Bitmap>(...) instead for better performance this.bitMaps = new HashMap<Integer, Bitmap>();
    this.selectedMap = new SparseBooleanArray();
    this.checkIds = new ArrayList<String>();
    unselectAll();
}
```

SparseArray是android里为<Integer,Object>这样的Hashmap而专门写的类,目的是提高效率。具体如何提高效率可以去Android文档查询一下,这里就不赘述了。

(二) 图片错位问题

这个问题导致的原因是因为复用convertView导致的,在加载大量的Item时,常见的错位问题。这种问题的解决思路通常是以图片的Url做为唯一的key,然后setTag中,然后获取时根据图片的URL来获得图片。

(三) 防止OOM,以及异步加载。

关于异步加载图片的思路是：

- 1.第一次进入时,是没有图片的,这时候我们会启动一个线程池,异步的从网上获得图片数据,为了防止图片过大导致OOM,可以调用BitmapFactory中的Options类对图片进行适当的缩放,最后再显示主线程的ImageView上。
- 2.把加载好的图片以图片的Url做为唯一的key存入内存缓存当中,并严格的控制好这个缓存的大小,防止OOM的发生。
- 3.把图片缓存在SD当中,如果没有SD卡就放在系统的缓存目录cache中,以保证在APP退出后,下次进来能看到缓存中的图片,这样就可以让你的APP不会给客户呈现一片空白的景象。
- 4.用户第二次进来的时候,加载图片的流程则是倒序的,首先从内容中看是否存在缓存图片,如果没有就从SD卡当中寻找,再没有然后才是从网络中获取图片数据。这样做的既可以提高加载图片的效率,同时也节约了用户的流量。

说完了理论性的东西,我们来开始动手实战一下吧,下面介绍一个GitHub上一个很轻巧的开源框架LazyList[GitHub地址](#),然后基于它做一些我们想要的效果,关于开源的东西,我们不止要学会用,还要从中能学到东西。众所周知的Android-Universal-Image-Loader其实就是基于LazyList的一个拓展,增加了更多的配置。但是从学习的角度我们还是希望能从原理学起,太多的功能封装,难免会让我们晕乎,简单的功能实现就够了。

- 1.先来看一下运行效果图：



2.来看一下LazyList项目的结构：

```
▷ FileCache.java 文件缓存类
▷ ImageLoader.java 图片加载的类，主要的
▷ LazyAdapter.java ListView的适配器
▷ MainActivity.java
▷ MemoryCache.java 内存缓存的类
▷ Utils.java
```

构非常的简单，整个项目的体积才10多k，适合加入我们自己的项目当中去，研究起来也不会觉得难，因为都是最为核心的东西。

3.调用Lazylist的入口：

[java]

[java]

```
01. <span style="font-size:14px;"> adapter = new LazyAdapter(this, mStrings);
02. list.setAdapter(adapter); </span>
```

传入一个装满图片Url地址的字符串数组进去，然后在LazyAdapter中对ListView中的进行显示。

4.具体LazyAdapter中调用的代码是：

[java]

[java]

```
01. <span style="font-size:14px;"> public class LazyAdapter extends BaseAdapter {
02.
03.     private Activity activity;
04.     private String[] data;
05.     private static LayoutInflater inflater=null;
06.     public ImageLoader imageLoader;
```

```

07.
08.     public LazyAdapter(Activity a, String[] d) {
09.         activity = a;
10.         data=d;
11.         inflater = (LayoutInflater)activity.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
12.         imageLoader=new ImageLoader(activity.getApplicationContext());
13.     }
14.
15.     public int getCount() {
16.         return data.length;
17.     }
18.
19.     public Object getItem(int position) {
20.         return position;
21.     }
22.
23.     public long getItemId(int position) {
24.         return position;
25.     }
26.
27.     public View getView(int position, View convertView, ViewGroup parent) {
28.         View vi=convertView;
29.         if(convertView==null)
30.             vi = inflater.inflate(R.layout.item, null);
31.
32.         ImageView image=BaseViewHolder.get(vi, R.id.image);
33.         ImageView image2=BaseViewHolder.get(vi, R.id.image2);
34.         imageLoader.DisplayImage(data[position], image);
35.         imageLoader.DisplayImage(data[position], image2);
36.         return vi;
37.     }
38. } </span>

```

5.从上面我们可以看出来其实最重要的封装显示图片的方法就在ImageLoader这个类中。

[java]

[java]

```

01. <span style="font-size:14px;">     public class ImageLoader {
02.
03.         MemoryCache memoryCache=new MemoryCache();
04.         FileCache fileCache;
05.         private Map<ImageView, String> imageViews=Collections.synchronizedMap(new WeakHashMap<ImageView, String>());
06.         ExecutorService executorService;
07.         Handler handler=new Handler();//handler to display images in UI thread
08.
09.         public ImageLoader(Context context){
10.             fileCache=new FileCache(context);
11.             executorService=Executors.newFixedThreadPool(5);
12.         }
13.         // 当进入listview时默认的图片，可换成你自己的默认图片
14.         final int stub_id=R.drawable.stub;
15.         public void DisplayImage(String url, ImageView imageView)
16.         {
17.             imageViews.put(imageView, url);
18.             // 先从内存缓存中查找
19.             Bitmap bitmap=memoryCache.get(url);
20.             if(bitmap!=null)
21.                 imageView.setImageBitmap(bitmap);
22.             else
23.             {
24.                 // 若没有的话则开启新线程加载图片
25.                 queuePhoto(url, imageView);
26.                 imageView.setImageResource(stub_id);
27.             }
28.         }
29.
30.         private void queuePhoto(String url, ImageView imageView)
31.         {
32.             PhotoToLoad p=new PhotoToLoad(url, imageView);

```

```

33.         executorService.submit(new PhotosLoader(p));
34.     }
35.
36.     private Bitmap getBitmap(String url)
37.     {
38.         File f=fileCache.getFile(url);
39.         /**
40.          * 先从文件缓存中查找是否有
41.          */
42.         //from SD cache
43.         Bitmap b = decodeFile(f);
44.         if(b!=null)
45.             return b;
46.         /**
47.          * 最后从指定的url中下载图片
48.          */
49.         //from web
50.         try {
51.             Bitmap bitmap=null;
52.             URL imageUrl = new URL(url);
53.             HttpURLConnection conn = (HttpURLConnection)imageUrl.openConnection();
54.             conn.setConnectTimeout(30000);
55.             conn.setReadTimeout(30000);
56.             conn.setInstanceFollowRedirects(true);
57.             InputStream is=conn.getInputStream();
58.             OutputStream os = new FileOutputStream(f);
59.             Utils.CopyStream(is, os);
60.             os.close();
61.             conn.disconnect();
62.             bitmap = decodeFile(f);
63.             return bitmap;
64.         } catch (Throwable ex){
65.             ex.printStackTrace();
66.             if(ex instanceof OutOfMemoryError)
67.                 memoryCache.clear();
68.             return null;
69.         }
70.     }
71.     /**
72.      * decode这个图片并且按比例缩放以减少内存消耗，虚拟机对每张图片的缓存大小也是有限的
73.      */
74.     //decodes image and scales it to reduce memory consumption
75.     private Bitmap decodeFile(File f){
76.         try {
77.             //decode image size
78.             BitmapFactory.Options o = new BitmapFactory.Options();
79.             o.inJustDecodeBounds = true;
80.             FileInputStream stream1=new FileInputStream(f);
81.             BitmapFactory.decodeStream(stream1,null,o);
82.             stream1.close();
83.
84.             //Find the correct scale value. It should be the power of 2.
85.             final int REQUIRED_SIZE=70;
86.             int width_tmp=o.outWidth, height_tmp=o.outHeight;
87.             int scale=1;
88.             while(true){
89.                 if(width_tmp/2<REQUIRED_SIZE || height_tmp/2<REQUIRED_SIZE)
90.                     break;
91.                 width_tmp/=2;
92.                 height_tmp/=2;
93.                 scale*=2;
94.             }
95.
96.             //decode with inSampleSize
97.             BitmapFactory.Options o2 = new BitmapFactory.Options();
98.             o2.inSampleSize=scale;
99.             FileInputStream stream2=new FileInputStream(f);
100.             Bitmap bitmap=BitmapFactory.decodeStream(stream2, null, o2);
101.             stream2.close();
102.             return bitmap;
103.         } catch (FileNotFoundException e) {
104.         }
105.         catch (IOException e) {
106.             e.printStackTrace();
107.         }
108.         return null;
109.     }
110.
111.     //Task for the queue

```

```

112.     private class PhotoToLoad
113.     {
114.         public String url;
115.         public ImageView imageView;
116.         public PhotoToLoad(String u, ImageView i){
117.             url=u;
118.             imageView=i;
119.         }
120.     }
121.
122.     class PhotosLoader implements Runnable {
123.         PhotoToLoad photoToLoad;
124.         PhotosLoader(PhotoToLoad photoToLoad){
125.             this.photoToLoad=photoToLoad;
126.         }
127.
128.         @Override
129.         public void run() {
130.             try{
131.                 if(imageViewReused(photoToLoad))
132.                     return;
133.                 Bitmap bmp=getBitmap(photoToLoad.url);
134.                 memoryCache.put(photoToLoad.url, bmp);
135.                 if(imageViewReused(photoToLoad))
136.                     return;
137.                 BitmapDisplayer bd=new BitmapDisplayer(bmp, photoToLoad);
138.                 handler.post(bd);
139.             }catch(Throwable th){
140.                 th.printStackTrace();
141.             }
142.         }
143.     }
144.     /**
145.      * 防止图片错位
146.      * @param photoToLoad
147.      * @return
148.      */
149.     boolean imageViewReused(PhotoToLoad photoToLoad){
150.         String tag=imageView.get(photoToLoad.imageView);
151.         if(tag==null || !tag.equals(photoToLoad.url))
152.             return true;
153.         return false;
154.     }
155.     /**
156.      * 用于在UI线程中更新界面
157.      *
158.      */
159.     //Used to display bitmap in the UI thread
160.     class BitmapDisplayer implements Runnable
161.     {
162.         Bitmap bitmap;
163.         PhotoToLoad photoToLoad;
164.         public BitmapDisplayer(Bitmap b, PhotoToLoad p){
165.             bitmap=b;
166.             photoToLoad=p;
167.         }
168.         public void run()
169.         {
170.             if(imageViewReused(photoToLoad))
171.                 return;
172.             if(bitmap!=null)
173.                 photoToLoad.imageView.setImageBitmap(bitmap);
174.             else
175.                 photoToLoad.imageView.setImageResource(stub_id);
176.         }
177.     }
178.
179.     public void clearCache() {
180.         memoryCache.clear();
181.         fileCache.clear();
182.     }
183.
184. }

```

由于篇幅的原因，FileCache和MenoryCache这两个缓存类，我就不贴大量代码了，全部都在我提供的demo示例代码中，你可以下，并有详细的注释，大家下载就可以用在自己的项目中了。

ListView优化的demo地址：<http://download.csdn.net/detail/finddreams/8141689>

原文:<http://blog.csdn.NET/finddreams/article/details/40977451>

顶 0 踩 0

- 上一篇 Android之极光推送发送自定义消息
- 下一篇 Android之实现定位

相关文章推荐

- 详解ListView加载网络图片的优化，让你轻松掌握！
- Android之ListView异步加载网络图片(优化缓存机...
- 优化ListView中的网络图片加载
- ListView异步加载网络图片之双缓存技术
- ListView AsyncTask异步加载网络Json格式数据和...
- ListView异步加载网络图片完美版之双缓存技术（...
- Android中ListView使用- 网络图片的异步加载
- Android Listview异步动态加载网络图片
- ListView使用SimpleAdapter加载网络图片
- 关于ListView中性能优化中图片加载问题



猜你在找

- 机器学习之概率与统计推断
- 机器学习之凸优化
- 响应式布局全新探索
- 深度学习基础与TensorFlow实践
- 前端开发在线峰会
- 机器学习之数学基础
- 机器学习之矩阵
- 探究Linux的总线、设备、驱动模型
- 深度学习之神经网络原理与实战技巧
- TensorFlow实战进阶：手把手教你做图像识别应用

查看评论

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场