自己造轮子：一款实用的Android广告栏实现过程

2016-01-23 安卓应用频道 安卓应用频道

(点击上方公众号，可快速关注)

来源：dongjunkun

链接：http://www.jianshu.com/p/bf3bc1f2df5c

开源界有一句很有名的话叫“不要重复发明轮子”，当然，我今天的观点不是要反驳这句话，轮子理论给我们的开发带来了极大的便利，项目中要实现一些功能，便去网上找找，一般推荐使用一些有名的库，我本身也是这么做的，但我想说的是，既要会用轮子，也要知道轮子怎么造，必要的时候，自己也要造轮子（想要找到一个完全满意的轮子还是不大容易的）。

由来

之前项目里面都是用的daimajia的AndroidImageSlider,一开始被惊艳的动画切换效果吸引了，还有各种自定义属性动画啥的，感觉很棒，但随着项目的进展和时间的推移，我慢慢发现它也不是无所不能的，甚至我发现它只是好看，却不怎么实用，我列举一些我发现的问题：

- 库中集成了Picasso，Picasso是极其耗内存的，这点我在之前的一篇为什么图片加载我首先Glide提到过，当一个页面既有广告又有列表（列表中也一般有图片）会造成页面严重卡顿。
- 动画效果太炫，实际项目中一个没用到，一般用标准模式。
- 太多的依赖包，现在的许多项目都是基于android4.0以上版本开发的，nineoldandroid已经不需要了
- 页面点击事件不直观，如果有那种像ListView的OnItemClickListener就好了

总而言之，我感觉它现在已经有些臃肿。当然，我也在网上寻找更简洁实用的替代库，比如BGABanner-Android,但事实也是残酷的，不支持网络图片，还有一个坑等着我跳，看代码片段

```
if (mAutoPlayAble && views.size() < 3) {
    throw new IllegalArgumentException("开启指定轮播时至少有三个页面");
}
```

低于3张图片会直接抛异常，这叫我怎么用，需求也不是我能控制的。
当然以上两个库的作者我都很喜欢，这两个库也非常不错，不然也不值得我花时间研究。

我对轮子的要求

结合自己的理解，我认为两个库中都有可取之处，也有不足之处，我就取长补短，造自己的轮子，我对这个轮子的要求：

- 无限轮播
- 自动加手动滑动
- 简单的自定义指示器样式及位置
- 支持本地图片及网络图片
- 滑动流畅，无卡顿，无闪烁
- 广告页面不限制个数
- 页面点击监听事件
- 简单易用，高配置，无明显bug
- 动画效果我先打算抛弃了，默认的就好，以实用为主。

开始动手，step by step

系统可以滑动翻页的控件就只有ViewPager和ViewFliper网上还有大神实现用RecyclerView实现了类似ViewPager的效果，这里暂不做过多研究，这里就选择使用最多ViewPager作为滑动翻页控件，使用ViewPager+PagerAdapter可以很容易的实现翻页切换效果，但存在几个弊端：

- 不能无限轮回的翻页（滑到第一个或者最后一个就不能继续滑）
- 切换速度太快，系统默认250毫秒，用做广告栏切换会存在明显闪烁
- 仅支持手动滑动，不支持自动切换
- 没有提供直接的类似ListView的OnItemClickListener监听，使用起来很不方便
- 当然还会有其他的坑，遇到了再解决，先解决上面的问题

无限轮播效果

这里采用网上通用的解决办法，伪轮播（让用户看到轮播的假象，实际上用了很多页面在不断重复出现，如果用户滑动几十亿下是可以滑到头，实际几乎不可能有人这么做）看具体实现代码

```
public class LoopPagerAdapter extends PagerAdapter {
    private List<View> views;

    public LoopPagerAdapter(List<View> views) {
        this.views = views;
    }

    @Override
    public int getCount() {
        //Integer.MAX_VALUE = 2147483647
        return Integer.MAX_VALUE;
    }

    @Override
    public boolean isViewFromObject(View view, Object object) {
```

```
        return view == object;
    }

    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        if (views.size() > 0) {
            //position % view.size()是指虚拟的position会在[0，view.size()）之间循环
            View view = views.get(position % views.size());
            if (container.equals(view.getParent())) {
                container.removeView(view);
            }
            container.addView(view);
            return view;
        }
        return null;
    }

    @Override
    public void destroyItem(ViewGroup container, int position, Object object) {
    }
}
```

还需要做一件事情，就是设置当前position为中间的一个较大的值，如果不设置或者设置的比较小，往左滑动容易滑倒头

```
pager.setCurrentItem(Integer.MAX_VALUE / 2 - Integer.MAX_VALUE / 2 % views.size());
```

改变原生ViewPager切换速度

通过反射拿到ViewPager的滑动器mScroller，改变duration参数，看代码：

```
public void setSliderTransformDuration(int duration) {
    try {
        Field mScroller = ViewPager.class.getDeclaredField("mScroller");
        mScroller.setAccessible(true);
        FixedSpeedScroller scroller = new FixedSpeedScroller(pager.getContext(), null, duration);
        mScroller.set(pager, scroller);
    } catch (Exception e) {

    }
}

//FixedSpeedScroller.java
public class FixedSpeedScroller extends Scroller {
  //默认1秒，可以通过上面的方法控制
   private int mDuration = 1000;
```

```
    public FixedSpeedScroller(Context context) {
        super(context);
    }

    public FixedSpeedScroller(Context context, Interpolator interpolator) {
        super(context, interpolator);
    }

    public FixedSpeedScroller(Context context, Interpolator interpolator, int duration){
        this(context,interpolator);
        mDuration = duration;
    }

    @Override
    public void startScroll(int startX, int startY, int dx, int dy, int duration) {
        // Ignore received duration, use fixed one instead
        super.startScroll(startX, startY, dx, dy, mDuration);
    }

    @Override
    public void startScroll(int startX, int startY, int dx, int dy) {
        // Ignore received duration, use fixed one instead
        super.startScroll(startX, startY, dx, dy, mDuration);
    }
}
```

自动切换实现

这里可以有多种方式，使用Handler或者Timer都可以的，这里采用handler实现，isAutoPlay可以控制是否禁止控件自动轮播，autoPlayDuration是轮播间隔时间，还需注意触摸时应当停止轮播，放开恢复正常

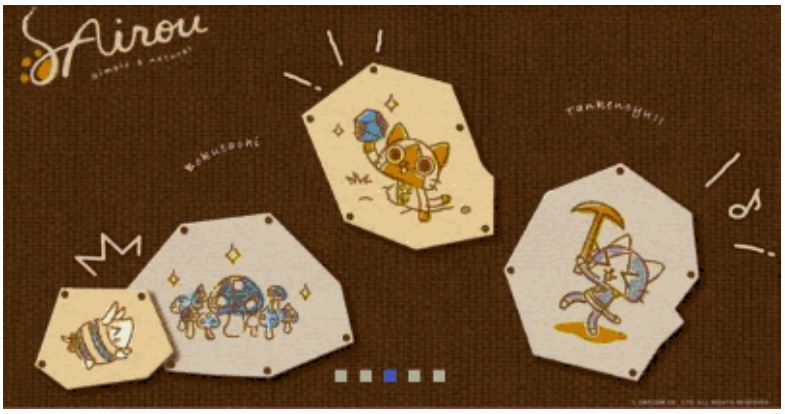
```
/**
     * 开始自动轮播
     */
    public void startAutoPlay() {
        if (isAutoPlay) {
            handler.sendEmptyMessageDelayed(WHAT_AUTO_PLAY, autoPlayDuration);
        }
    }

    /**
     * 停止自动轮播
     */
    public void stopAutoPlay() {
        if (isAutoPlay) {
            handler.removeMessages(WHAT_AUTO_PLAY);
        }
```

```
    }

    @Override
    public boolean dispatchTouchEvent(MotionEvent ev) {
        switch (ev.getAction()) {
            case MotionEvent.ACTION_DOWN:
                stopAutoPlay();
                break;
            case MotionEvent.ACTION_CANCEL:
            case MotionEvent.ACTION_UP:
                startAutoPlay();
                break;
        }
        return super.dispatchTouchEvent(ev);
    }
```

先附上一张预览图：



先写这么多吧，后续完整代码我会上传到github，如果大家有兴趣，我会抽时间写剩下的内容，这个控件的代码也是借鉴了很多优秀的开源库，并结合自己的理解写的。

> 炫丽的效果固然吸引人眼球，平凡实用的东西才愈久弥香

完整代码已上传到github:BannerLayoutDemo

添加指示器–绘制指示器样式

github上也有各种各样很棒的指示器，可作为独立控件，这里我先简单处理直接集成到内部，后期有需求再进行重构，最简单的指示器是用两张不同状态的小图片做的，但我认为这样做相对实现是简单的，但对于修改却显得有些麻烦，适配也是问题，简单修改何必大动干戈呢？

这里借鉴了daimajia的思路，指示器是用代码绘出来的，怎么绘呢？看代码，以选中的状态为例，绘制一次便可，将drawable存起来使用：

```
    Drawable selectedDrawable;
```

```
GradientDrawable selectedGradientDrawable = new GradientDrawable();
//设置指示器的颜色
selectedGradientDrawable.setColor(selectedIndicatorColor);
//设置指示器的形状
selectedGradientDrawable.setShape(GradientDrawable.RECTANGLE);
//设置指示器的大小
selectedGradientDrawable.setSize(selectedIndicatorWidth, selectedIndicatorHeight);
selectedLayerDrawable = new LayerDrawable(new Drawable[]{selectedGradientDrawable});
selectedDrawable = selectedLayerDrawable;
```

这样指示器的形状，大小，颜色可以随意换了，支持换肤也更容易

添加指示器–位置

大家都知道在xml文件中使用RelativeLayout父布局可以控制子布局的位置，用代码怎么去做呢？首先，BannerLayout是继承与RelativeLayout的，看代码

```
RelativeLayout.LayoutParams params = new LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT);
 switch (indicatorPosition) {
    case centerBottom://下中
        params.addRule(RelativeLayout.CENTER_HORIZONTAL);
        params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
        break;
    case rightBottom://右下
        params.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);
        params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM);
        break;
 }
//添加指示器容器布局到BannerLayout
addView(indicatorContainer, params);
```

两个属性必须分开添加不能写成

```
params.addRule(RelativeLayout.ALIGN_PARENT_RIGHT|RelativeLayout.ALIGN_PARENT_BOTTOM);
```

如何设置指示器margin和padding这里就不再多说

切换指示器

我之前的想法是这样的，不用循环做，只需知道上一个选中的页面和即将要跳转的页面位置即可，实现思路是这样的

```
private void switchIndicator(int currentPosition) {
```

```
        if (oldPosition != -1){
            ((ImageView)indicatorContainer.getChildAt(oldPosition)).setImageDrawable(unSelectedDrawable);
        }
        ((ImageView)indicatorContainer.getChildAt(currentPosition)).setImageDrawable(selectedDrawable);
        oldPosition = currentPosition;
    }
```

但实际的运行效果却可能出现错乱的现象，不知是哪里出了问题，目前就采用了循环来做，后期可能会改进，这样做虽然简单，但效率始终不高

```
    private void switchIndicator(int currentPosition) {
        for (int i = 0; i < indicatorContainer.getChildCount(); i++) {
            ((ImageView) indicatorContainer.getChildAt(i)).setImageDrawable(i == currentPosition ? selectedDrawable
    : unSelectedDrawable);
        }
    }
```

添加页面点击监听回调

用法就像ListView的setOnItemClickListener一样，来看看代码如何实现

```
    private OnBannerItemClickListener onBannerItemClickListener;
    //给每个页面添加点击事件
    imageView.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                if (onBannerItemClickListener != null) {
                    //不直接处理点击事件，转交给onBannerItemClickListener
                    onBannerItemClickListener.onItemClick(position);
                }
            }
        });
    public void setOnBannerItemClickListener(OnBannerItemClickListener onBannerItemClickListener) {
        this.onBannerItemClickListener = onBannerItemClickListener;
    }
    public interface OnBannerItemClickListener {
        void onItemClick(int position);
    }
```

使用

```
    bannerLayout.setOnBannerItemClickListener(new BannerLayout.OnBannerItemClickListener() {
            @Override
            public void onItemClick(int position) {
                //处理点击事件
            }
```

```
        });
```

遇到的一些坑

上篇讲到了BGABanner-Android,部分思想也是参考这个库的，例如指示器位置的处理方案，但我要说的坑也在这里，低于3张图片就会直接抛异常，我试着将异常不抛出看看，一张或者两张图片的时候切换效果惨不忍睹（我猜想和ViewPager的懒加载机制有关），所以作者处理为直接抛异常，但这样不行啊，需求不可控制啊，必须解决这个问题，不然就像定时炸弹。

问题解决思路：既然轮播是伪的，图片的张数也可以是伪的，只需要给用户看起来是那样就行了，1张也可以是3×1张相同的图片，2张也可以是2×2张相同图片，3张及以上没问题就无需处理，关键代码

```
    //添加本地图片路径
    public void setViewRes(List<Integer> viewRes) {
        List<View> views = new ArrayList<>();
        itemCount = viewRes.size();
        //主要是解决当item为小于3个的时候滑动有问题，这里将其拼凑成3个以上
        if (itemCount < 1) {//当item个数0
            throw new IllegalStateException("item count not equal zero");
        } else if (itemCount < 2) {//当item个数为1
            views.add(getImageView(viewRes.get(0), 0));
            views.add(getImageView(viewRes.get(0), 0));
            views.add(getImageView(viewRes.get(0), 0));
        } else if (itemCount < 3) {//当item个数为2
            views.add(getImageView(viewRes.get(0), 0));
            views.add(getImageView(viewRes.get(1), 1));
            views.add(getImageView(viewRes.get(0), 0));
            views.add(getImageView(viewRes.get(1), 1));
        } else {
            for (int i = 0; i < viewRes.size(); i++) {
                views.add(getImageView(viewRes.get(i), i));
            }
        }
        setViews(views);
    }
```

添加网络地址和本地的思路差不多，使用的是Glide来处理加载网络图片，还有就是添加各种自定义属性。

github地址BannerLayoutDemo。

https://github.com/dongjunkun/BannerLayoutDemo

完整代码实现请看源码，欢迎fork和star以及提出你宝贵的意见