链滴

Android-ViewPager

作者：wind

原文链接：https://ld246.com/article/1441016599084

来源网站：链滴

1. ViewPager使用示例
2. 源码解析

示例
ViewPager可以说是IM中必用的一个控件了，举例比如WeChat，Weibo，其用法也非常之简单。

<pre class="prettyprint lang-xml">   <android.support.v4.view.ViewPager
        android:id="@+id/viewpager"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
    </android.support.v4.view.ViewPager></pre>

xml文件中定义完之后，再看java文件中代码

<pre class="prettyprint lang-java">public class MainActivity extends AppCompatActivity {

private ViewPager viewpager=null;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    viewpager=(ViewPager)findViewById(R.id.viewpager);
    MyAdapter adapter=new MyAdapter();
    LayoutInflater lf=getLayoutInflater();
    adapter.addviews(lf.inflate(R.layout.page1,null));
    adapter.addviews(lf.inflate(R.layout.page2,null));
    adapter.addviews(lf.inflate(R.layout.page3,null));
    viewpager.setAdapter(adapter);
}

}</pre>

可以看到ViewPager的用法和ListView的用法是类似的，都需要一个Adapter来加载数据，下面看这自定义的MyAdapter。

<pre class="prettyprint lang-java">public class MyAdapter extends PagerAdapter {

private ArrayList<View> views;

public void addviews(View v){

```
    views.add(v);
}
@Override
public int getCount() {
    return views.size();
}


@Override
public boolean isViewFromObject(View view, Object object) {
    return view==object;
}


@Override
public void destroyItem(ViewGroup container, int position, Object object)   {
    container.removeView(views.get(position));
}


@Override
public Object instantiateItem(ViewGroup container,int position){
    container.addView(views.get(position));
    return views.get(position);
}

}</pre>

<p>
    <br />
</p>
这个MyAdapter继承PagerAdapter，可以看到，需要Override四个方法，方法作用可以从函数名得。<br />
很简单吧？跟ListView的用法真是太相似了，ViewPager的内部还有一个接口<br />
<pre class="prettyprint lang-java">public interface OnPageChangeListener {
   public void onPageScrolled(int position, float positionOffset, int positionOffsetPixels);
   public void onPageSelected(int position);
   public void onPageScrollStateChanged(int state);
}</pre>
<br />
<p>
    接口的作用也很明显，最常用的就是用来加载各个tab的内容。
</p>
<p>
    <br />
</p>
<p>
    网上的用法有很多，这里就不细讲了，重点是下一部分。
</p>
<p>
    <br />
</p>
<span style="font-size:16px;">源码</span><br />
开始看到ViewPager的滑动效果时，马上联想到了ListView的下拉刷新，想到ViewPager也是采用这
```

原理来实现的，后来一看源码，果然，其原理就是相当于将多个View平行的摆放在一起，滑动即时调横向的offset，来实现滑屏效果。
初始化的代码跳过，直接看onTouchEvent函数的代码，
首先介绍一下ViewPager中比较重要的一些变量和类。
<pre class="prettyprint lang-java">private final ArrayList<ItemInfo> mItems = new Arra
List<ItemInfo>();
static class ItemInfo {
        Object object;   //View或者Image
        int position;   //在mItems中的position，一般等同于offset
        boolean scrolling;
        float widthFactor;
        float offset;
    }</pre>

mItems是一个ItemInfo的ArrayList，很重要，是前一个Page，当前Page，后一个Page的集合。

ACTION_DOWN
<pre class="prettyprint lang-java">case MotionEvent.ACTION_DOWN: {
                mScroller.abortAnimation();
                mPopulatePending = false;
                populate();

            // Remember where the motion event started
            mLastMotionX = mInitialMotionX = ev.getX();
            mLastMotionY = mInitialMotionY = ev.getY();
            mActivePointerId = MotionEventCompat.getPointerId(ev, 0);
            break;
        }</pre>

重点是记录X，Y，PointerId，和populate()函数，前面的变量都好理解，这个populate函数还是比复杂的，我们看看。

<pre class="prettyprint lang-java"> final int pageLimit = mOffscreenPageLimit;//偏离屏幕pag
限制，一般左右各一个
        final int startPos = Math.max(0, mCurItem - pageLimit);左边的page
        final int N = mAdapter.getCount();
        final int endPos = Math.min(N-1, mCurItem + pageLimit);右边的page
        .....
        int curIndex = -1;
        ItemInfo curItem = null;
        for (curIndex = 0; curIndex < mItems.size(); curIndex++) {
            final ItemInfo ii = mItems.get(curIndex);
            if (ii.position >= mCurItem) {
                if (ii.position == mCurItem) curItem = ii;
                break;
            }
        }   //找到curItem
        .....
        if (curItem != null) {
            float extraWidthLeft = 0.f;
            int itemIndex = curIndex - 1;
            ItemInfo ii = itemIndex >= 0 ? mItems.get(itemIndex) : null;

```
    final int clientWidth = getClientWidth();
    final float leftWidthNeeded = clientWidth &lt;= 0 ? 0 :
            2.f - curItem.widthFactor + (float) getPaddingLeft() / (float) clientWidth;
    for (int pos = mCurItem - 1; pos &gt;= 0; pos--) {
        if (extraWidthLeft &gt;= leftWidthNeeded &amp;&amp; pos &lt; startPos) {
            if (ii == null) {
                break;
            }
            if (pos == ii.position &amp;&amp; !ii.scrolling) {
                mItems.remove(itemIndex);
                mAdapter.destroyItem(this, pos, ii.object);
                if (DEBUG) {
                    Log.i(TAG, "populate() - destroyItem() with pos: " + pos +
                            " view: " + ((View) ii.object));
                }
                itemIndex--;
                curIndex--;
                ii = itemIndex &gt;= 0 ? mItems.get(itemIndex) : null;
            }
        } else if (ii != null &amp;&amp; pos == ii.position) {
            extraWidthLeft += ii.widthFactor;
            itemIndex--;
            ii = itemIndex &gt;= 0 ? mItems.get(itemIndex) : null;
        } else {
            ii = addNewItem(pos, itemIndex + 1);
            extraWidthLeft += ii.widthFactor;
            curIndex++;
            ii = itemIndex &gt;= 0 ? mItems.get(itemIndex) : null;
        }
    }


float extraWidthRight = curItem.widthFactor;
itemIndex = curIndex + 1;
if (extraWidthRight &lt; 2.f) {
    ii = itemIndex &lt; mItems.size() ? mItems.get(itemIndex) : null;
    final float rightWidthNeeded = clientWidth &lt;= 0 ? 0 :
            (float) getPaddingRight() / (float) clientWidth + 2.f;
    for (int pos = mCurItem + 1; pos &lt; N; pos++) {
        if (extraWidthRight &gt;= rightWidthNeeded &amp;&amp; pos &gt; endPos) {
            if (ii == null) {
                break;
            }
            if (pos == ii.position &amp;&amp; !ii.scrolling) {
                mItems.remove(itemIndex);
                mAdapter.destroyItem(this, pos, ii.object);
                if (DEBUG) {
                    Log.i(TAG, "populate() - destroyItem() with pos: " + pos +
                            " view: " + ((View) ii.object));
                }
                ii = itemIndex &lt; mItems.size() ? mItems.get(itemIndex) : null;
            }
        } else if (ii != null &amp;&amp; pos == ii.position) {
            extraWidthRight += ii.widthFactor;
```

```
                itemIndex++;
                ii = itemIndex &lt; mItems.size() ? mItems.get(itemIndex) : null;
            } else {
                ii = addNewItem(pos, itemIndex);
                itemIndex++;
                extraWidthRight += ii.widthFactor;
                ii = itemIndex &lt; mItems.size() ? mItems.get(itemIndex) : null;
            }
        }
    }</pre>
```


这是核心代码，代码有点长，主要作用就是产生一个mItems，使得mItems中含有左右以及当前的Pa
e，产生的依据是extraWidth和WidthNeeded的关系，这里注意，mItems有多个地方会产生调用，
如onMeasure中，所以一次touch滑屏，会分别调用多次populate()，一般调用两次吧。
extraWidth和WidthNeeded分Left和Right，现只拿Right来做讲解。
首先float extraWidthRight=curItem.widthFactor;
widthFactor宽度因子，一般为1，代表一个page的宽度。
final float rightWidthNeeded=(float)getPaddingRight()/(float)clientWidth+2.f;
rightWidthNeeded=2,很明显，不用计算。
进入for循环，先假设总共有3个Page，现在从2滑到1
先看第2次调用吧，即屏幕已经从2变成了1，但是mItems里的值还没有发生改变，但是curItem是1
。
pos=mCurItem+1=1，此时pos=1,endPos=1，ii代表的是itemIndex所对应的mItem，现在是pag
2，进入else if语句，extraWidthRight+=ii.widthFactor，此时extraWidthRight=2，即curItem和
边Page的宽度总和。

    itemIndex++;得到page3所对应的mItem，pos=2>endPos，直接进入if语句，我们可以看到

<pre class="prettyprint lang-java"> mItems.remove(itemIndex);
 mAdapter.destroyItem(this, pos, ii.object);</pre>

成功删除了page3这个对象，mItems的大小变为了2。
以上就是ViewPager的核心--mItems，基本由populate掌控。
后面的就比较简单了，上面的代码看了比较久，主要是在观察各种width和mItems之间的关系，还有
万不要忽略了滑屏过后onMeasure方法会重新调用populate()。

    ACTION_MOVE

<pre class="prettyprint lang-java"> if (mIsBeingDragged) {
                final int activePointerIndex = MotionEventCompat.findPointerIndex(
                        ev, mActivePointerId);

```
                final float x = MotionEventCompat.getX(ev, activePointerIndex);
                needsInvalidate |= performDrag(x);
            }</pre>
<br />
<p>
  核心函数是performDrag(x)，作用顾名思义，就是产生drag的效果。
</p>
<p>
  <br />
</p>
   
<pre class="prettyprint lang-java"> private boolean performDrag(float x) {
        boolean needsInvalidate = false;


    final float deltaX = mLastMotionX - x;
    mLastMotionX = x;


    float oldScrollX = getScrollX();
    float scrollX = oldScrollX + deltaX;
    final int width = getClientWidth();


    float leftBound = width * mFirstOffset;
    float rightBound = width * mLastOffset;
    boolean leftAbsolute = true;
    boolean rightAbsolute = true;


    final ItemInfo firstItem = mItems.get(0);
    final ItemInfo lastItem = mItems.get(mItems.size() - 1);
    if (firstItem.position != 0) {
        leftAbsolute = false;
        leftBound = firstItem.offset * width;
    }
    if (lastItem.position != mAdapter.getCount() - 1) {
        rightAbsolute = false;
        rightBound = lastItem.offset * width;
    }


    if (scrollX &lt; leftBound) {
        if (leftAbsolute) {
            float over = leftBound - scrollX;
            needsInvalidate = mLeftEdge.onPull(Math.abs(over) / width);
        }
        scrollX = leftBound;
    } else if (scrollX &gt; rightBound) {
        if (rightAbsolute) {
            float over = scrollX - rightBound;
            needsInvalidate = mRightEdge.onPull(Math.abs(over) / width);
        }
        scrollX = rightBound;
```

```
    }
    // Don't lose the rounded component
    mLastMotionX += scrollX - (int) scrollX;
    scrollTo((int) scrollX, getScrollY());
    pageScrolled((int) scrollX);


    return needsInvalidate;
}

</pre>
<br />
别看代码很乱，主要就是leftBound，RightBound，leftAbsolute，rightAbsolute的关系，这里还注意First和Last，千万不要以为是第一个Page和最后一个Page，这个是相对于mItems来说的，所以面的mItems概念理解错了，这里也会理解错。<br />
leftAbsolute=true表示，左边没有Page了，RightAbsolute也一样。<br />
从上到下的这些if分别是判断，firstItem和lastItem是否在边缘，socrllX的判断，则是在leftAbsolut或者RightAbsolute为true时起作用，即是超过边缘，无效的滑动。最后scrollTo实现drag效果。<br>
<strong>ACTION_UP</strong><br />
其实弹起后的代码就很好想了，无非是根据fling这类的手势或者offset来判断是否要跳到下一个page代码也很简单了，大家自己看看吧。<br />
<br />
<br />
其实我们也可以看到，android系统自带的ViewPager中mItems的使用是非常节约内存的。为了锻炼己对View和ViewGroup的理解，决定下星期自己写一个ScrollLayou试试。<br />
<br />
<br />
<br />
<br />
<br />
```