宝贵建议请发送至：wangzhenyang@itcast.cn

黑马程序员 itheima.com -编程，始于黑马

# Android 课程同步笔记

Beta 0.01 版

By 阳哥

# Android 自定义控件-2ListView 下拉刷新&加载更多&黑马新闻

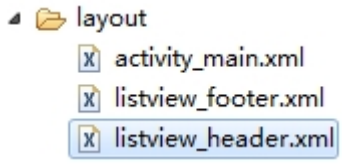
## 1. ListView 下拉刷新&分页加载（★★）

### 1.1 简介


如上图所示为本文档中要实现的效果，这也是我们在使用 Android 手机时经常见到的效果。当往下拉 ListView 的时候在 ListView 头部添加一个视图，当往上拉 ListView 的时候在 ListView 脚部添加一个视图。

### 1.2 布局实现

我们的演示工程共用到三个布局文件，分别是主 Activity 布局、ListView 头布局、ListView 脚布局。布局文件名如下图所示：




activity_main.xml 文件清单如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <com.itheima.refreshlistviewdemo.view.RefreshListView
        android:id="@+id/refresh_listview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >
    </com.itheima.refreshlistviewdemo.view.RefreshListView>
</RelativeLayout>
```

Tips：该布局非常的简单，只有一个自定义的 ListView 对象。

listview_header.xml 文件清单如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <FrameLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dip" >

        <ImageView
            android:id="@+id/iv_listview_header_arrow"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:src="@drawable/common_listview_headview_red_arrow" />

        <ProgressBar
            android:id="@+id/pb_listview_header"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
```

```
            android:indeterminateDrawable="@drawable/custom_progressbar"
            android:visibility="invisible" />
    </FrameLayout>

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:gravity="center_horizontal"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/tv_listview_header_state"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="下拉刷新"
            android:textColor="#FF0000"
            android:textSize="20sp" />

        <TextView
            android:id="@+id/tv_listview_header_last_update_time"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="5dip"
            android:text="最后刷新时间: 1990-09-09 09:98:09"
            android:textColor="@android:color/darker_gray"
            android:textSize="16sp" />
    </LinearLayout>

</LinearLayout>
```

listview_footer.xml 文件清单如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:orientation="horizontal" >

    <ProgressBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_margin="5dip"
        android:indeterminateDrawable="@drawable/custom_progressbar" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dip"
        android:text="加载更多中..."
        android:textColor="#FF0000"
        android:textSize="25sp" />

</LinearLayout>
```

# 1.3 业务逻辑实现

## 1.3.1 自定义 ListView 对象

因为我们使用的 ListView 需要添加上头部视图和脚部视图，因此 Android 原生的 ListView 并不能满足我们的需求，这时我们可以自定义一个 ListView 对象，并继承原生 ListView 对象。

RefreshListView.java 代码清单如下：

```
/**
 * 自定义 ListView 的下拉刷新头和加载更多尾
 */
public class RefreshListView extends ListView implements OnScrollListener {

    private int downY; // 按下时 y 轴的偏移量
    private int headerViewHeight; // 头布局的高度
    private View headerView; // 头布局对象

    private final int DOWN_PULL = 0;    // 头布局状态：下拉刷新
    private final int RELEASE_REFRESH = 1;// 头布局状态：  释放刷新
    private final int REFRESHING = 2;  // 头布局状态：  正在刷新中..

    private int currentState = DOWN_PULL; // 头布局当前的状态，默认为：下拉刷新
    private RotateAnimation upAnimation;  // 头布局向上旋转的动画
    private RotateAnimation downAnimation; // 头布局向下旋转的动画
```

```
    private ImageView ivArrow; // 头布局的箭头
    private ProgressBar mProgressBar; // 头布局的进度圈
    private TextView tvState; // 头布局的状态
    private TextView tvLastUpdateTime; // 头布局最后刷新时间

    private OnRefreshListener mOnRefreshListener;   // 使用者的回调事件
    private View footerView; // 脚布局对象
    private int footerViewHeight; // 脚布局的高度
    private boolean isLoadingMore = false; // 是否正在加载更多中，默认为：没有正在加载

    public RefreshListView(Context context) {
        super(context);
        initHeaderView();
        initFooterView();
        setOnScrollListener(this);
    }

    public RefreshListView(Context context, AttributeSet attrs) {
        super(context, attrs);
        initHeaderView();
        initFooterView();
        setOnScrollListener(this);
    }

    /**
     * 初始化脚布局
     */
    private void initFooterView() {
        footerView = View.inflate(getContext(), R.layout.listview_footer, null);

        // 设置脚布局的 paddingTop 为自己高度的负数
        footerView.measure(0, 0);
        footerViewHeight = footerView.getMeasuredHeight();
        footerView.setPadding(0, -footerViewHeight, 0, 0);

        this.addFooterView(footerView);
    }

    /**
     * 初始化 ListView 下拉刷新头
     */
    private void initHeaderView() {
        headerView = View.inflate(getContext(), R.layout.listview_header, null);
```

```
        ivArrow = (ImageView)
headerView.findViewById(R.id.iv_listview_header_arrow);
        mProgressBar = (ProgressBar)
headerView.findViewById(R.id.pb_listview_header);
        tvState = (TextView) headerView.findViewById(R.id.tv_listview_header_state);
        tvLastUpdateTime = (TextView)
headerView.findViewById(R.id.tv_listview_header_last_update_time);

        tvLastUpdateTime.setText("最后刷新时间: " + getCurrentTime());

        // 测量头布局的高度.
        headerView.measure(0, 0); // 让系统框架去帮我们测量头布局的宽和高.

        // 取出头布局的高度.
//      headerView.getHeight(); // 此方法是控件没有显示到屏幕上之前是获取不到值的, 一直
都是 0
        headerViewHeight = headerView.getMeasuredHeight(); // 获得一个测量后的高度, 只
有在 measure 方法被调用完毕后才可以得到具体高度.
        System.out.println("头布局的高度: " + headerViewHeight);

        // 隐藏头布局. paddingTop
        headerView.setPadding(0, -headerViewHeight, 0, 0);

        // 向 ListView 的顶部追加一个布局.
        this.addHeaderView(headerView);

        initAnimation();
    }

    /**
     * 初始化动画
     */
    private void initAnimation() {
        upAnimation = new RotateAnimation(
                0, -180,
                Animation.RELATIVE_TO_SELF, 0.5f,
                Animation.RELATIVE_TO_SELF, 0.5f);
        upAnimation.setDuration(500);
        upAnimation.setFillAfter(true); // 让控件停止在动画结束的状态下

        downAnimation = new RotateAnimation(
                -180, -360,
                Animation.RELATIVE_TO_SELF, 0.5f,
```

```
                Animation.RELATIVE_TO_SELF, 0.5f);
        downAnimation.setDuration(500);
        downAnimation.setFillAfter(true); // 让控件停止在动画结束的状态下
    }

    @Override
    public boolean onTouchEvent(MotionEvent ev) {
        switch (ev.getAction()) {
        case MotionEvent.ACTION_DOWN:
            downY = (int) ev.getY();
            break;
        case MotionEvent.ACTION_MOVE:
            // 当前的状态是否是正在刷新中，如果是，直接跳出.
            if(currentState == REFRESHING) {
                break;
            }

            int moveY = (int) ev.getY();

            // 间距 = 移动 y - 按下 y;
            int diffY = moveY - downY;
            // 计算头布局最新的 paddingTop = -头布局高度 + 间距.
            int paddingTop = -headerViewHeight + diffY;
//          System.out.println("paddingTop: " + paddingTop);

            // 如果 paddingTop 的值 < -headerViewHeight，不进行下拉刷新头的滑动操作.
            // 并且 ListView 顶部第一个显示的条目的索引为: 0，才可以进行滑动.

            // 获取 ListView 顶部第一个显示的条目的索引
            int firstVisiblePosition = getFirstVisiblePosition();
//          System.out.println("firstVisiblePosition: " + firstVisiblePosition);
            if(paddingTop > -headerViewHeight
                    && firstVisiblePosition == 0) {

                if(paddingTop > 0 && currentState == DOWN_PULL) { // 头布局完全显示，并且当前状态是下拉刷新，进入到松开刷新的状态
                    System.out.println("松开刷新");
                    currentState = RELEASE_REFRESH;
                    refreshHeaderViewState();
                } else if(paddingTop < 0 && currentState == RELEASE_REFRESH) { // 头布局没有完全显示，并且当前状态是松开刷新，进入到下拉刷新的状态
                    System.out.println("下拉刷新");
                    currentState = DOWN_PULL;
```

```
                refreshHeaderViewState();
            }
            headerView.setPadding(0, paddingTop, 0, 0);
            return true; // 自己处理用户触摸滑动的事件.
        }
        break;
    case MotionEvent.ACTION_UP:
        // 判断当前的状态是哪一种
        if(currentState == DOWN_PULL) { // 当前是在下拉刷新状态下松开了，什么都不做，把头布局隐藏就可以.
            headerView.setPadding(0, -headerViewHeight, 0, 0);
        } else if(currentState == RELEASE_REFRESH) { // 当前的状态属于释放刷新，并且松开了. 应该把头布局正常显示，进入正在刷新中状态.
            headerView.setPadding(0, 0, 0, 0);
            currentState = REFRESHING;
            refreshHeaderViewState();

            // 调用用户的监听事件.
            if(mOnRefreshListener != null) {
                mOnRefreshListener.onPullDownRefresh();
            }
        }
        break;
    default:
        break;
    }
    return super.onTouchEvent(ev); // ListView 默认的滑动效果.
}

/**
 * 根据当前的状态 currentState 来刷新头布局的状态.
 */
private void refreshHeaderViewState() {
    switch (currentState) {
    case DOWN_PULL:  // 下拉刷新
        ivArrow.startAnimation(downAnimation);
        tvState.setText("下拉刷新");
        break;
    case RELEASE_REFRESH: // 松开刷新
        ivArrow.startAnimation(upAnimation);
        tvState.setText("松开刷新");
        break;
    case REFRESHING: // 正在刷新中
```

```
            ivArrow.clearAnimation(); // 把自己身上的动画清除掉
            ivArrow.setVisibility(View.INVISIBLE);
            mProgressBar.setVisibility(View.VISIBLE);
            tvState.setText("正在刷新..");
            break;
        default:
            break;
        }
    }

    /**
     * 刷新完成，用户调用此方法，把对应的头布局或脚布局给隐藏掉
     */
    public void onRefreshFinish() {
        if(isLoadingMore) { // 当前属于加载更多中
            // 隐藏脚布局
            footerView.setPadding(0, -footerViewHeight, 0, 0);
            isLoadingMore = false;

        } else { // 下拉刷新操作
            // 隐藏头布局
            headerView.setPadding(0, -headerViewHeight, 0, 0);
            currentState = DOWN_PULL;
            mProgressBar.setVisibility(View.INVISIBLE);
            ivArrow.setVisibility(View.VISIBLE);
            tvState.setText("下拉刷新");
            tvLastUpdateTime.setText("最后刷新时间: " + getCurrentTime());
        }
    }

    /**
     * 获取最新的时间
     * @return 1990-09-09 09:09:09
     */
    private String getCurrentTime() {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        return sdf.format(new Date());
    }

    /**
     * 提供给使用者设置刷新的监听事件
     * @param listener
     */
```

```
    public void setOnRefreshListener(OnRefreshListener listener) {
        mOnRefreshListener = listener;
    }

    /**
     * 当 ListView 刷新时的监听事件
     */
    public interface OnRefreshListener {

        /**
         * 当下拉刷新时回调此方法
         */
        public void onPullDownRefresh();

        /**
         * 当加载更多时调用此方法
         */
        public void onLoadingMore();
    }

    @Override
    public void onScroll(AbsListView view, int firstVisibleItem,
            int visibleItemCount, int totalItemCount) {

    }

    /**
     * 当滚动状态改变时，触发此方法.
     * scrollState 当前滚动的状态.
     *
     * OnScrollListener.SCROLL_STATE_IDLE; 停滞状态
     * OnScrollListener.SCROLL_STATE_TOUCH_SCROLL; 手指触摸在屏幕上滑动.
     * OnScrollListener.SCROLL_STATE_FLING; 手指快速的滑动一下.
     *
     */
    @Override
    public void onScrollStateChanged(AbsListView view, int scrollState) {
        // 当前的状态是停止，并且屏幕上显示的最后一个条目的索引是 ListView 中总条目个数
-1;
//     System.out.println("scrollState: " + scrollState + ", last: " +
getLastVisiblePosition() + ", count: " + getCount());
        if((scrollState == OnScrollListener.SCROLL_STATE_IDLE // 当前是停滞或者是快速
滑动时
```

```
            || scrollState == OnScrollListener.SCROLL_STATE_FLING)
            && getLastVisiblePosition() == (getCount() -1)
            && !isLoadingMore) {
          System.out.println("滑动到底部，可以加载更多数据了.");

          isLoadingMore = true;
          footerView.setPadding(0, 0, 0, 0);
          setSelection(getCount()); // 滑动到最底部

          if(mOnRefreshListener != null) {
             mOnRefreshListener.onLoadingMore();
          }
       }
    }
}
```

## 1.3.2 MainActivity 中使用自定义的 ListView 对象

```
public class MainActivity extends Activity {

   private List<String> dataList;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final RefreshListView mListView = (RefreshListView)
findViewById(R.id.refresh_listview);
        dataList = new ArrayList<String>();
        for (int i = 0; i < 30; i++) {
          dataList.add("这是 ListView 的数据: " + i);
       }

        // 给 ListView 设置 Adapter 数据适配器
        final MyAdapter mAdapter = new MyAdapter();
        mListView.setAdapter(mAdapter);

        // 设置一个当 ListView 刷新的监听
        mListView.setOnRefreshListener(new OnRefreshListener() {
```

```
        @Override
        public void onPullDownRefresh() {
            Toast.makeText(MainActivity.this, "开始下拉刷新了", 0).show();

            new Handler().postDelayed(new Runnable() {
                @Override
                public void run() {
                    // 会在 3 秒钟后执行.
                    dataList.add(0, "我是下拉刷新出来的数据..");
                    mAdapter.notifyDataSetChanged();

                    // 把头布局隐藏掉
                    mListView.onRefreshFinish();
                }
            }, 3000);
        }

        @Override
        public void onLoadingMore() {
            Toast.makeText(MainActivity.this, "开始加载更多了", 0).show();

            new Handler().postDelayed(new Runnable() {
                @Override
                public void run() {
                    dataList.add("我是加载更多的数据 1");
                    dataList.add("我是加载更多的数据 2");
                    dataList.add("我是加载更多的数据 3");
                    mListView.onRefreshFinish();
                }
            }, 5000);
        }
    });
}

/**
 * 数据适配器
 */
class MyAdapter extends BaseAdapter {

    @Override
    public int getCount() {
        return dataList.size();
    }
```

```
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            TextView tv = new TextView(MainActivity.this);
            tv.setText(dataList.get(position));
            tv.setTextSize(18);
            tv.setTextColor(Color.BLACK);
            tv.setPadding(0, 5, 0, 5);
            return tv;
        }

        @Override
        public Object getItem(int position) {
            return null;
        }

        @Override
        public long getItemId(int position) {
            return 0;
        }

    }
}
```

Tips：类似我们做的 RefreshListView 功能的开源框架有很多，在真实的开发中，讲究效率的条件下很多时候我们都会直接使用开源框架。推荐 GitHub 网站，上面可以搜索各种各样的开源项目。

# 2. 黑马新闻（★★）

## 2.1 简介

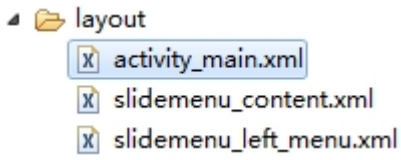
黑马新闻主要练习的是用代码实现类似 SlidingMenu（滑动菜单）的功能。

效果图如下所示：

左图为黑马新闻首页，向右滑动屏幕后打开左侧菜单，效果为右图。


## 2.2 布局实现

该项目总共使用到了三个布局文件，如下图所示。




activity_main.xml 文件清单

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <com.itheima.slidingmenudemo.view.SlideMenu
        android:id="@+id/slidemenu"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <!-- 引入菜单布局，索引为：0 -->
        <include layout="@layout/slidemenu_left_menu" />

        <!-- 引入主界面布局，索引为：1 -->
        <include layout="@layout/slidemenu_content" />

    </com.itheima.slidingmenudemo.view.SlideMenu>

</RelativeLayout>
```

Tips：这个布局文件用到了自定义的 ViewGroup 对象 com.itheima.slidingmenudemo.view.SlideMenu，继承了 ViewGroup 类的控件时可以在其内部嵌套子控件。

slidemenu_content.xml 文件清单

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/top_bar_bg"
        android:orientation="horizontal" >

        <ImageButton
            android:id="@+id/ib_back"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="@android:color/transparent"
            android:src="@drawable/main_back" />

        <View
            android:layout_width="1dip"
            android:layout_height="fill_parent"
            android:layout_margin="5dip"
```

```
            android:background="@drawable/top_bar_divider" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:layout_marginLeft="10dip"
            android:text="黑马新闻"
            android:textColor="#FFFFFF"
            android:textSize="30sp" />
    </LinearLayout>

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="center"
        android:text="钓鱼岛是中国的!!\nxxx 是世界的."
        android:textColor="#000000"
        android:textSize="30sp" />

</LinearLayout>
```

Tips：该文件为黑马新闻主界面布局。

slidemenu_left_menu.xml 文件清单

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="240dip"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="240dip"
        android:layout_height="match_parent"
        android:background="@drawable/menu_bg"
        android:orientation="vertical" >

        <TextView
            style="@style/left_menu_tab"
            android:background="#33663300"
            android:drawableLeft="@drawable/tab_news"
            android:text="新闻" />
```

```
        <TextView
            style="@style/left_menu_tab"
            android:drawableLeft="@drawable/tab_read"
            android:text="订阅" />

        <TextView
            style="@style/left_menu_tab"
            android:drawableLeft="@drawable/tab_local"
            android:text="本地" />

        <TextView
            style="@style/left_menu_tab"
            android:drawableLeft="@drawable/tab_ties"
            android:text="跟帖" />

        <TextView
            style="@style/left_menu_tab"
            android:drawableLeft="@drawable/tab_pics"
            android:text="图片" />

        <TextView
            style="@style/left_menu_tab"
            android:drawableLeft="@drawable/tab_ugc"
            android:text="话题" />

        <TextView
            style="@style/left_menu_tab"
            android:drawableLeft="@drawable/tab_vote"
            android:text="投票" />

        <TextView
            style="@style/left_menu_tab"
            android:drawableLeft="@drawable/tab_focus"
            android:text="聚合阅读" />
    </LinearLayout>

</ScrollView>
```

# 2.3 业务逻辑实现

## 2.3.2 知识点清单

- 自定义 ViewGroup 控件
- Android 事件处理机制

## 2.3.3 代码实现

自定义类 SlideMenu 继承 ViewGroup。该类作为自定义控件类。

- SlideMenu.java 代码清单

```
public class SlideMenu extends ViewGroup {

    private int downX; // 按下时 x 轴的偏移量
    private final int SCREEN_MENU = 0; // 菜单界面
    private final int SCREEN_MAIN = 1; // 主界面

    private int currentScreen = SCREEN_MAIN; // 当前屏幕显示的界面，默认为：主界面

    private Scroller scroller;
    private int touchSlop;

    public SlideMenu(Context context, AttributeSet attrs) {
        super(context, attrs);
        init();
    }

    public SlideMenu(Context context) {
        super(context);
        init();
    }

    private void init() {
        scroller = new Scroller(getContext());

        touchSlop = ViewConfiguration.get(getContext()).getTouchSlop();
    }
```

```
    /**
     * widthMeasureSpec 填充屏幕
     * heightMeasureSpec 填充屏幕
     */
    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);

        // 测量菜单的宽和高. 宽: 240dip, 高: 填充屏幕
        View menuView = getChildAt(0); // 获取菜单对象
        menuView.measure(menuView.getLayoutParams().width, heightMeasureSpec);

        // 测量主界面的宽和高. 宽: 填充屏幕, 高: 填充屏幕
        View mainView = getChildAt(1);
        mainView.measure(widthMeasureSpec, heightMeasureSpec);
    }

    /**
     * int l 左边=0
     * int t 上边=0
     * int r 右边=屏幕的宽度
     * int b 下边=屏幕的高度
     */
    @Override
    protected void onLayout(boolean changed, int l, int t, int r, int b) {
        // 主界面的位置放置在屏幕左上角
        View mainView = getChildAt(1);
        mainView.layout(l, t, r, b);

        // 把菜单的位置放置在屏幕的左侧
        View menuView = getChildAt(0);
        menuView.layout(-menuView.getMeasuredWidth(), t, 0, b);
    }

    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {
        // 只有在横着滑动时才可以拦截.
        switch (ev.getAction()) {
        case MotionEvent.ACTION_DOWN:
            downX = (int) ev.getX();
            break;
        case MotionEvent.ACTION_MOVE:
            int moveX = (int) ev.getX();
```

```
            int diff = Math.abs(downX - moveX);
            if(diff > touchSlop) {
                return true;
            }
            break;
        default:
            break;
        }

        return super.onInterceptTouchEvent(ev);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int scrollX;

        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            downX = (int) event.getX();
            break;
        case MotionEvent.ACTION_MOVE:
            int moveX = (int) event.getX();

            int deltaX = downX - moveX;

            // 判断给定当前的增量移动后，是否能够超出边界.
            scrollX = getScrollX() + deltaX;
            if(scrollX < -getChildAt(0).getMeasuredWidth()) {
                // 当前超出了左边界，应该设置为在菜单的左边界位置上.
                scrollTo(-getChildAt(0).getMeasuredWidth(), 0);
            } else if(scrollX > 0) {
                // 当前超出了右边界，应该设置为0
                scrollTo(0, 0);
            } else {
                scrollBy(deltaX, 0);
            }

            downX = moveX;
            break;
        case MotionEvent.ACTION_UP:
            // 获取菜单宽度的一半
            int center = -getChildAt(0).getMeasuredWidth() / 2;
            scrollX = getScrollX(); // 当前屏幕左上角的值
```

```
//        Toast.makeText(getContext(), "scrollX="+getScrollX(), 0).show();
          Toast.makeText(getContext(), "center =
"+center*2+"&layout.parameters="+getChildAt(0).getLayoutParams().width, 0).show();
          if(scrollX > center) {
             System.out.println("当前切换到主界面");
             currentScreen = SCREEN_MAIN;
          } else {
             System.out.println("当前切换到菜单界面");
             currentScreen = SCREEN_MENU;
          }

          switchScreen();
          break;
       default:
          break;
       }
       return true;
    }

    /**
     * 根据 currentScreen 变量来切换屏幕显示
     */
    private void switchScreen() {
       int startX = getScrollX(); // 开始的位置
       int dx = 0; // 增量值 = 目的地位置 - 开始的位置;

       if(currentScreen == SCREEN_MAIN) {
//        scrollTo(0, 0);
          dx = 0 - startX;
       } else if(currentScreen == SCREEN_MENU) {
//        scrollTo(-getChildAt(0).getMeasuredWidth(), 0);
          dx = -getChildAt(0).getMeasuredWidth() - startX;
       }

       int duration = Math.abs(dx) * 10;
       if(duration > 1000) {
          duration = 1000;
       }
       scroller.startScroll(startX, 0, dx, 0, duration);

       // 刷新当前控件，会引起 onDraw 方法的调用.
       invalidate(); // -> drawChild -> view.draw -> view.computeScroll
    }
```

```
    @Override
    public void computeScroll() {
        // 当 scroller 数据模拟完毕时，不应该继续进行递归
        // 反之，如果正在模拟数据才进行递归的操作

        if(scroller.computeScrollOffset()) {// 当前还是正在模拟数据中
            // 把当前 scroller 正在模拟的数据取出来，使用 scrollTo 方法切换屏幕
            int currX = scroller.getCurrX();
            scrollTo(currX, 0);
            invalidate(); // 在触发当前方法，相当于递归.
        }
    }

    /**
     * 是否显示菜单
     * @return true 显示菜单，false 不显示
     */
    public boolean isShowMenu() {
        return currentScreen == SCREEN_MENU;
    }

    /**
     * 隐藏菜单
     */
    public void hideMenu() {
        currentScreen = SCREEN_MAIN;
        switchScreen();
    }

    /**
     * 显示菜单
     */
    public void showMenu() {
        currentScreen = SCREEN_MENU;
        switchScreen();
    }
}
```

MainActivity.java 代码清单

MainActivity 比较简单，只需要使用我们自定义的控件即可。

```
public class MainActivity extends Activity implements OnClickListener {

    private SlideMenu mSlideMenu;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);

        mSlideMenu = (SlideMenu) findViewById(R.id.slidemenu);
        findViewById(R.id.ib_back).setOnClickListener(this);
    }

    public void clickTab(View v) {
        TextView tv = (TextView) v;
        Toast.makeText(this, tv.getText(), 0).show();
    }

    @Override
    public void onClick(View v) {
        // 切换菜单的显示或者隐藏的操作

        // 判断当前是哪一个屏幕
        boolean isShowMenu = mSlideMenu.isShowMenu();

        if(isShowMenu) {
            // 是菜单界面，切换到主界面
            mSlideMenu.hideMenu();
        } else {
            // 是主界面，应该把菜单显示出来
            mSlideMenu.showMenu();
        }
    }
}
```

***至此，本文档完！***

2015 年 3 月 1 日 星期日 15:16:54

北京市海淀区中关村软件园国际软件大厦