# **电子书翻页效果的实现**

目录

[电子书翻页效果的实现 1](#_Toc41726446)

[§0 关于报告的说明 4](#_Toc41726447)

[阅读建议 4](#_Toc41726448)

[心得体会 5](#_Toc41726449)

[不足 5](#_Toc41726450)

[§1导入项目 5](#_Toc41726451)

[§1.1 将项目名改成全英 5](#_Toc41726452)

[§1.2 .gradle版本不一致 5](#_Toc41726453)

[§1.3 字符编码问题 5](#_Toc41726454)

[§2代码结构 6](#_Toc41726455)

[§2.1 Activity起始活动 6](#_Toc41726456)

[§2.2 显示第一页 7](#_Toc41726457)

[§2.3 向前翻页 7](#_Toc41726458)

[§2.4 向后翻页 8](#_Toc41726459)

[§2.5 组件触摸事件 8](#_Toc41726460)

[§3 项目类详解 8](#_Toc41726461)

[§3.1 turntest类 8](#_Toc41726462)

[§3.1.1 摘要 9](#_Toc41726463)

[※属性 9](#_Toc41726464)

[※方法 9](#_Toc41726465)

[§3.1.2 属性 9](#_Toc41726466)

[※mPageWidget 9](#_Toc41726467)

[※mCurPageBitmap 9](#_Toc41726468)

[※mNextPageBitmap 9](#_Toc41726469)

[※mCurPageCanvas 9](#_Toc41726470)

[※mNextPageCanvas 10](#_Toc41726471)

[※pagefactory 10](#_Toc41726472)

[§3.1.3 方法 10](#_Toc41726473)

[※onCreate 10](#_Toc41726474)

[§3.2 PageWidget类 10](#_Toc41726475)

[§3.2.1 摘要 10](#_Toc41726476)

[※属性 10](#_Toc41726477)

[※方法 11](#_Toc41726478)

[§3.2.2 属性 12](#_Toc41726479)

[※mWidth 12](#_Toc41726480)

[※mHeight 12](#_Toc41726481)

[※mCornerX 12](#_Toc41726482)

[※mCornerY 12](#_Toc41726483)

[※mPath0 13](#_Toc41726484)

[※mPath1 13](#_Toc41726485)

[※mCurPageBitmap 13](#_Toc41726486)

[※mNextPageBitmap 13](#_Toc41726487)

[※mTouch 13](#_Toc41726488)

[※mBezierStart1 14](#_Toc41726489)

[※mBezierControl1 14](#_Toc41726490)

[※mBeziervertex1 15](#_Toc41726491)

[※mBezierEnd1 15](#_Toc41726492)

[※mBezierStart2 15](#_Toc41726493)

[※mBezierControl2 16](#_Toc41726494)

[※mBeziervertex2 16](#_Toc41726495)

[※mBezierEnd2 17](#_Toc41726496)

[※mMiddleX 17](#_Toc41726497)

[※mMiddleY 17](#_Toc41726498)

[※mDegrees 18](#_Toc41726499)

[※mTouchToCornerDis 18](#_Toc41726500)

[※mColorMatrixFilter 19](#_Toc41726501)

[※mMatrix 19](#_Toc41726502)

[※mMatrixArray 19](#_Toc41726503)

[※mIsRTandLB 19](#_Toc41726504)

[※mMaxLength 19](#_Toc41726505)

[※mBackShadowColors 19](#_Toc41726506)

[※mFrontShadowColors 19](#_Toc41726507)

[※mBackShadowDrawableLR 20](#_Toc41726508)

[※mBackShadowDrawableRL 20](#_Toc41726509)

[※mFolderShadowDrawableLR 20](#_Toc41726510)

[※mFolderShadowDrawableRL 20](#_Toc41726511)

[※mFrontShadowDrawableHBT 20](#_Toc41726512)

[※mFrontShadowDrawableHTB 20](#_Toc41726513)

[※mFrontShadowDrawableVLR 20](#_Toc41726514)

[※mFrontShadowDrawableVRL 21](#_Toc41726515)

[※mPaint 21](#_Toc41726516)

[※mScroller 21](#_Toc41726517)

[§3.2.3 方法 21](#_Toc41726518)

[※PageWidget 21](#_Toc41726519)

[※calcCornerXY 22](#_Toc41726520)

[※doTouchEvent 22](#_Toc41726521)

[※getCross 23](#_Toc41726522)

[※setBitmaps 24](#_Toc41726523)

[※drawCurrentPageShadow 24](#_Toc41726524)

[※computeScroll 27](#_Toc41726525)

[※abortAnimation 28](#_Toc41726526)

[※canDragOver 28](#_Toc41726527)

[※DragToRight 29](#_Toc41726528)

[※calcPoints 29](#_Toc41726529)

[※drawCurrentPageArea 31](#_Toc41726530)

[※drawNextPageAreaAndShadow 32](#_Toc41726531)

[※createDrawable 34](#_Toc41726532)

[※drawCurrentBackArea 35](#_Toc41726533)

[※startAnimation 36](#_Toc41726534)

[※onDraw 37](#_Toc41726535)

[§3.3 BookPageFactory类 37](#_Toc41726536)

[§3.3.1 摘要 37](#_Toc41726537)

[※属性 37](#_Toc41726538)

[※方法 38](#_Toc41726539)

[§3.3.2 属性 38](#_Toc41726540)

[※book\_file 38](#_Toc41726541)

[※m\_mbBuf 38](#_Toc41726542)

[※m\_mbBufLen 39](#_Toc41726543)

[※m\_mbBufBegin 39](#_Toc41726544)

[※m\_mbBufEnd 39](#_Toc41726545)

[※m\_strCharsetName 39](#_Toc41726546)

[※m\_book\_bg 39](#_Toc41726547)

[※mWidth 39](#_Toc41726548)

[※mHeight 39](#_Toc41726549)

[※m\_lines 40](#_Toc41726550)

[※m\_fontSize 40](#_Toc41726551)

[※m\_textColor 40](#_Toc41726552)

[※m\_backColor 40](#_Toc41726553)

[※marginWidth 40](#_Toc41726554)

[※marginHeight 40](#_Toc41726555)

[※mLineCount 40](#_Toc41726556)

[※mVisibleHeight 41](#_Toc41726557)

[※mVisibleWidth 41](#_Toc41726558)

[※m\_isfirstPage 41](#_Toc41726559)

[※m\_islastPage 41](#_Toc41726560)

[※mPaint 41](#_Toc41726561)

[§3.3.3 方法 41](#_Toc41726562)

[※BookPageFactory 41](#_Toc41726563)

[※openbook 42](#_Toc41726564)

[※nextPage 42](#_Toc41726565)

[※onDraw 43](#_Toc41726566)

[※setBgBitmap 44](#_Toc41726567)

[※isfirstPage 44](#_Toc41726568)

[※islastPage 44](#_Toc41726569)

[※readParagraphBack 44](#_Toc41726570)

[※readParagraphForward 46](#_Toc41726571)

[※pageDown 47](#_Toc41726572)

[※pageUp 48](#_Toc41726573)

[※prePage 49](#_Toc41726574)

[§4 其他类详解 49](#_Toc41726575)

[§4.1 Path类 49](#_Toc41726576)

[§4.2 Bitmap类 50](#_Toc41726577)

[§4.3 PointF类 50](#_Toc41726578)

[§4.4 Paint类 50](#_Toc41726579)

[§4.5 Scroller类 50](#_Toc41726580)

[§4.6 GradientDrawable类 51](#_Toc41726581)

[§4.7 MappedByteBuffer类 51](#_Toc41726582)

[§4.8 ColorMatrix类 52](#_Toc41726583)

[§4.9 ColorMatrixColorFilter类 52](#_Toc41726584)

[§4.10 RandomAccessFile类 52](#_Toc41726585)

[§4.11 DecimalFormat类 53](#_Toc41726586)

[§5 其他要点 53](#_Toc41726587)

[§5.1 安卓坐标系与视图坐标系 53](#_Toc41726588)

[§5.2 View类的要点 54](#_Toc41726589)

[§5.2.1 invalidate()和postInvalidate()区分 54](#_Toc41726590)

[§5.2.2 View的三大流程 54](#_Toc41726591)

[§5.2.3 onDraw()的调用时间 54](#_Toc41726592)

[§5.3 Canvas类的要点 55](#_Toc41726593)

[§5.4 computeScroll和Scroller滑动事件的处理 55](#_Toc41726594)

[§5.5 JAVA设计模式之工厂模式—Factory Pattern 56](#_Toc41726595)

[§5.6 贝塞尔曲线的理解 56](#_Toc41726596)

[§5.7 事件分发机制原理 56](#_Toc41726597)

[§5.8 绘图的几个要素 57](#_Toc41726598)

## §0 关于报告的说明

### 阅读建议

* 在word或者在wps中打开时，建议将缩放比例调整到大小合适的比例（在制作word时采用140%的比例）；
* 阅读时，最好将导航栏打开（word中，勾选“视图-显示-导航窗格”）；
* 文中有些地方有链接，按住ctrl键单击即可调转；
* 项目中要读取的txt文件路径已经修改，不再是sd卡中的，但是toast窗口中未找到文件的提示信息没有修改。

### 心得体会

虽然项目不大，但是经过本次上机阅读项目之后，积累了一定的代码经验，开阔了自己的视野。结合百度和帮助文档，使用了很多包中的类和很多方法，对安卓项目的配置、自定义组件的使用、事件处理机制、UI窗体等有了一个较为深刻的理解。

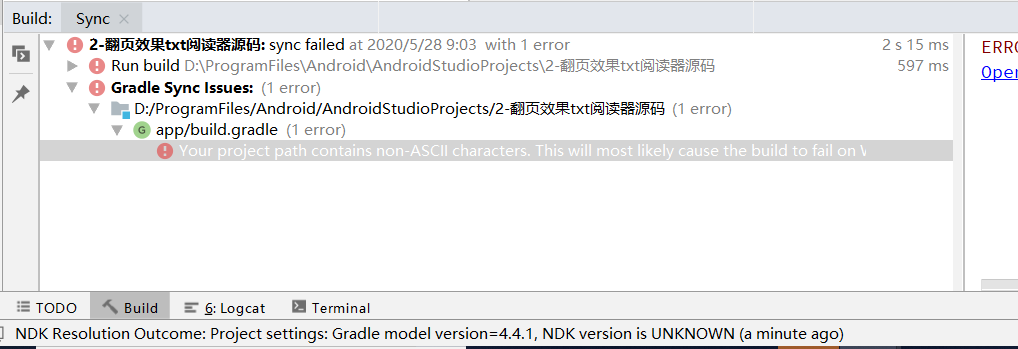
### 不足

* 文档中有一些地方加上超链接可能更加完美；
* 有些地方只要思想能够明白，但是有些细节似懂非懂；
* 项目太小。

## §1导入项目

### §1.1 将项目名改成全英

Sync failed.



ERROR: Your project path contains non-ASCII characters. This will most likely cause the build to fail on Windows. Please move your project to a different directory.

解决方案：

将项目名改为全英且不带特殊字符即可，同时保证所有路径中不包含中文或其他特殊字符。

### §1.2 .gradle版本不一致

在sync过程中自动下载对应的.gradle版本

### §1.3 字符编码问题

修改字符编码，andorid studio具有很智能的错误处理能力。

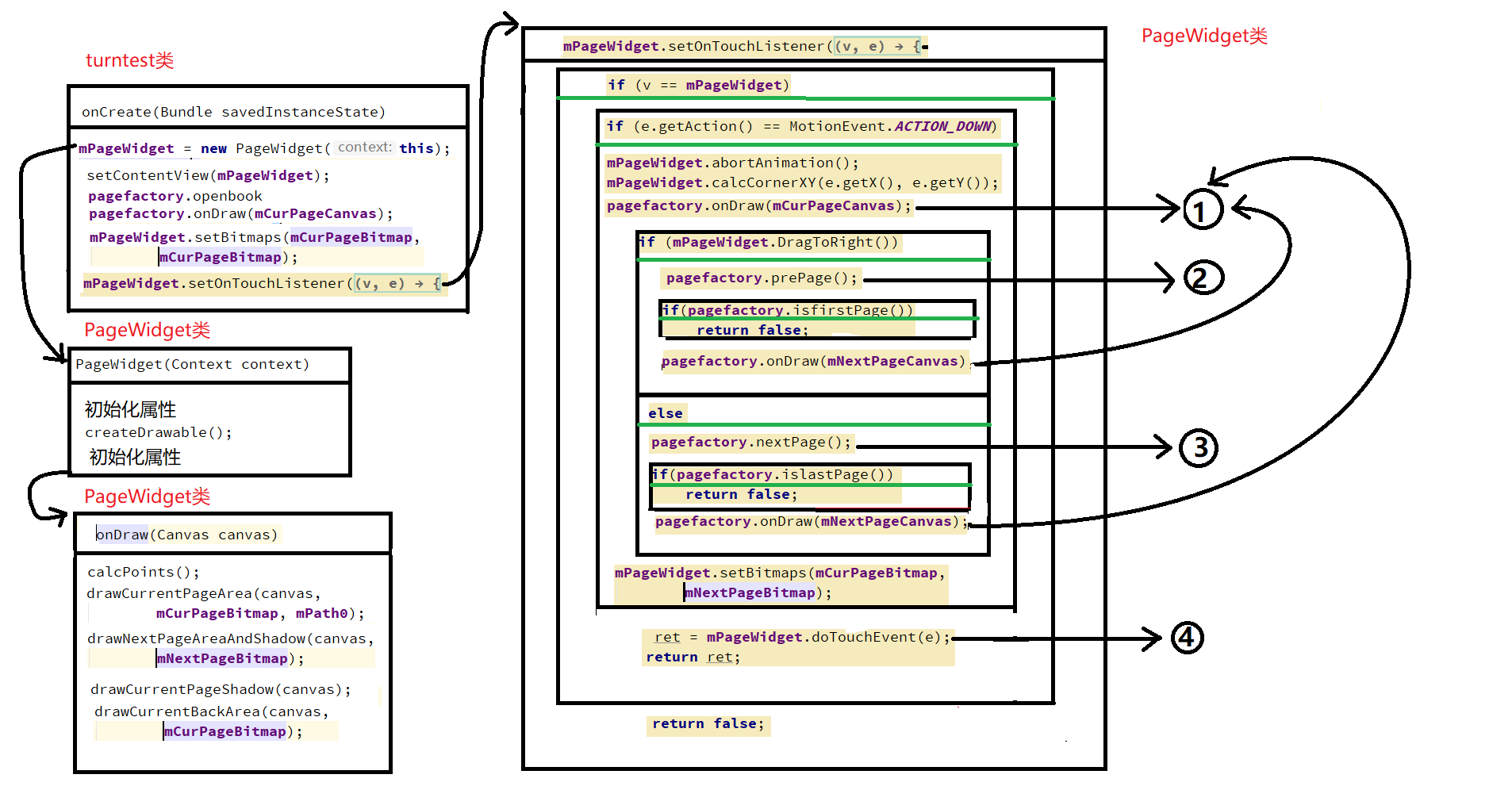
## §2代码结构

代码结构既可以从上到下连续地看成各函数依次调用情况，也可以分开看成每一块为某一种机制的具体实现。

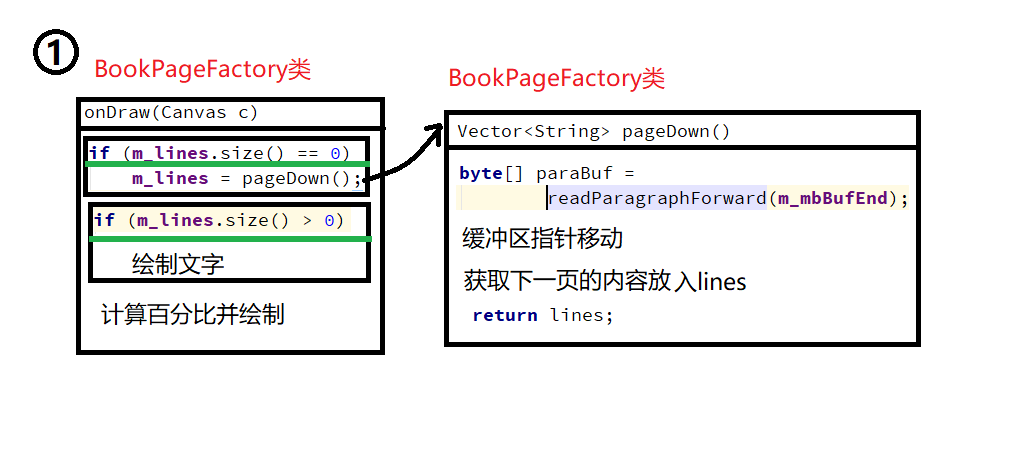
注：下列方框表示代码块，方框上方的小块为代码块的条件（头）：用**黑线**相隔的为方法的方法名与方法体 或者 if-else语句；用**绿线**相隔的为判断条件与满足该条件应该执行的代码块；**红字**说明该方法属于哪一个类。

### §2.1 Activity起始活动

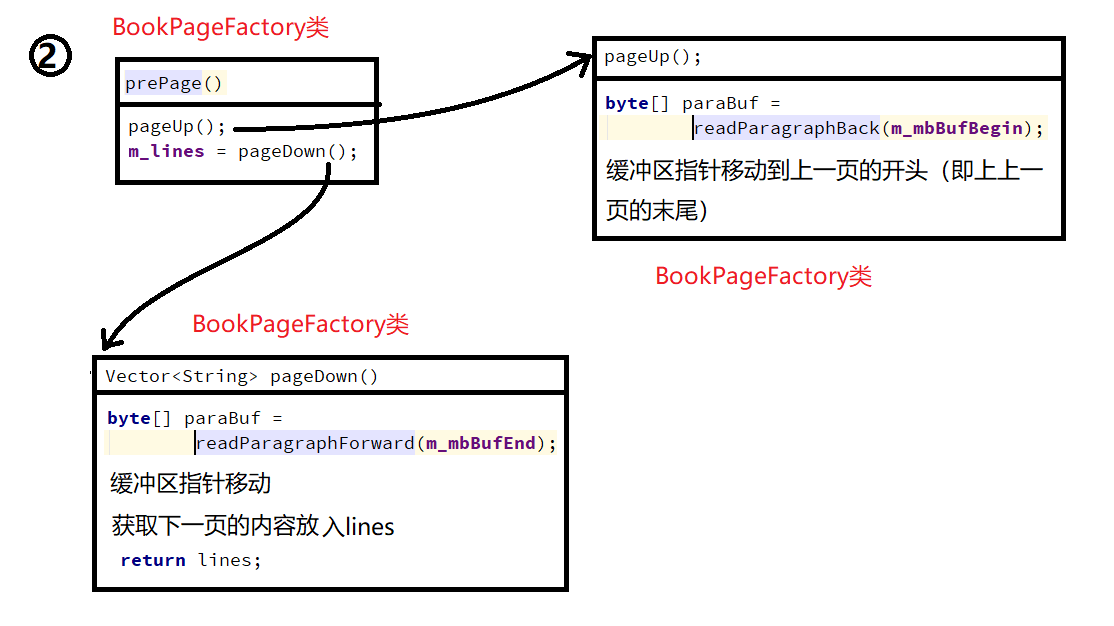
从Activity开始，设置手指触摸的监听事件，通过手指触摸的动作调用相应的方法实现翻页的绘制与文字的绘制。



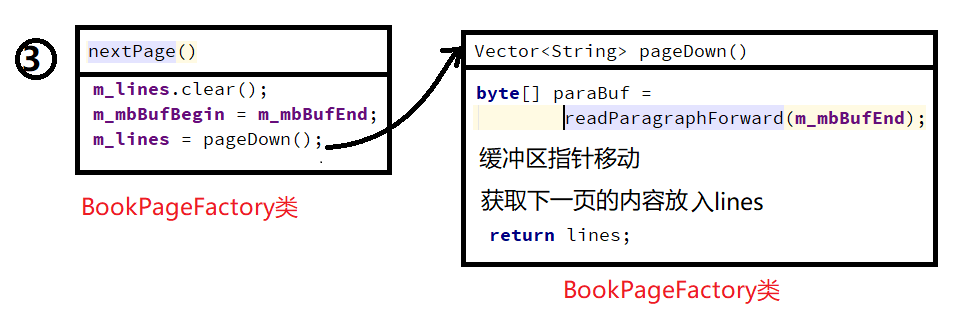
### §2.2 显示第一页



### §2.3 向前翻页

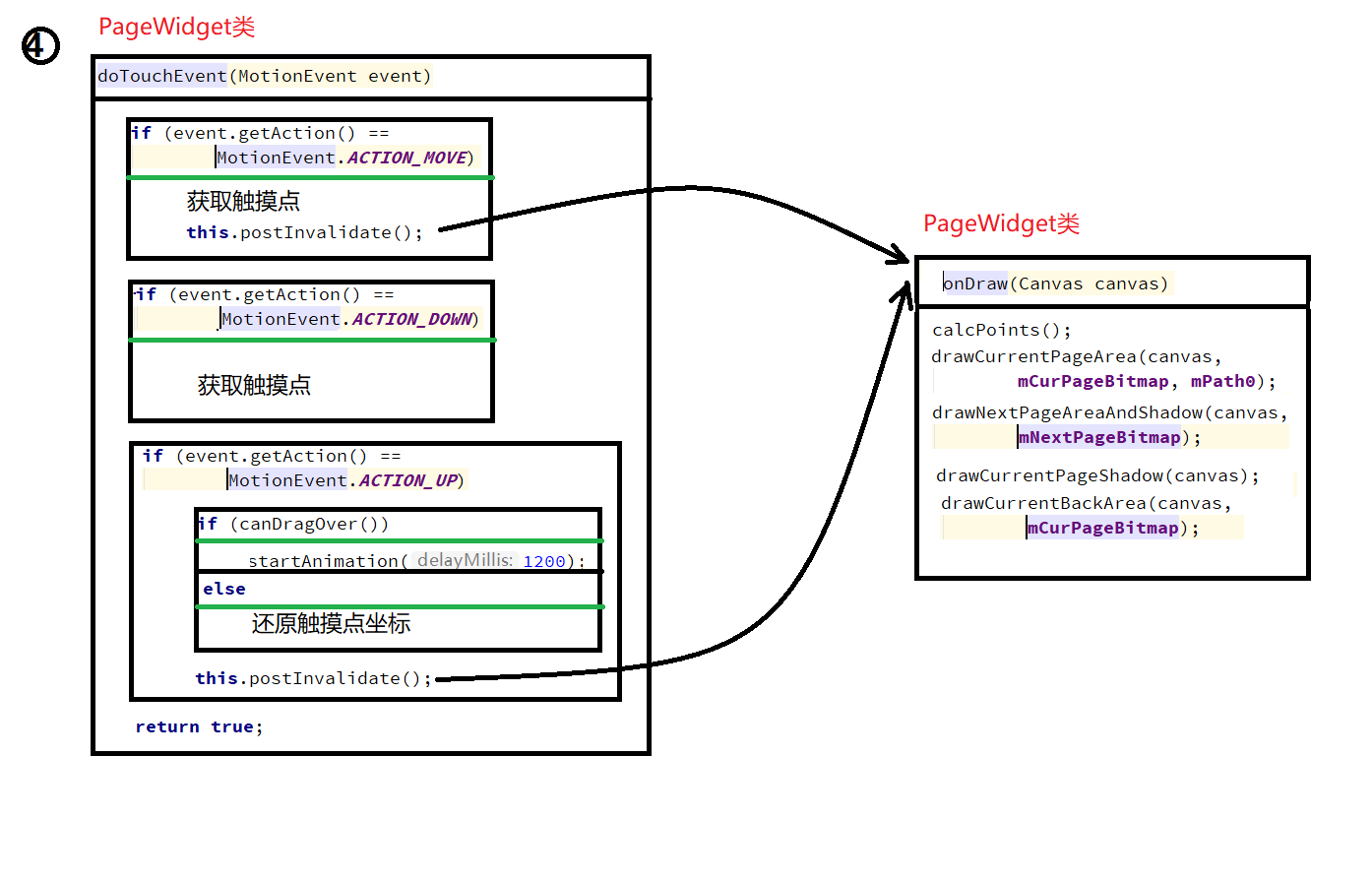


### §2.4 向后翻页



### §2.5 组件触摸事件

[View类的onDraw(Canvas canvas)方法的调用时间。](#_§5.2.3_onDraw()的调用时间)



## §3 项目类详解

### §3.1 turntest类

**public class** turntest **extends** Activity

#### §3.1.1 摘要

##### ※属性

**private** PageWidget **mPageWidget**

Bitmap **mCurPageBitmap**

Bitmap **mNextPageBitmap**

Canvas **mCurPageCanvas**

Canvas **mNextPageCanvas**

BookPageFactory **pagefactory**

##### ※方法

**public void** onCreate(Bundle savedInstanceState)

#### §3.1.2 属性

##### ※mPageWidget

类型：**private** [PageWidget](#_§3.2_PageWidget类)

自定义组件。

##### ※mCurPageBitmap

类型：[Bitmap](#_§4.2_Bitmap类)

当前页的位图。

##### ※mNextPageBitmap

类型：[Bitmap](#_§4.2_Bitmap类)

下一页的位图。

##### ※mCurPageCanvas

类型：Canvas

当前页的画布。

##### ※mNextPageCanvas

类型：Canvas

下一页的画布。

##### ※pagefactory

类型：[BookPageFactory](#_§3.3_BookPageFactory类)

按页读取文件的工厂类实例。

#### §3.1.3 方法

##### ※onCreate

**public void** onCreate(Bundle savedInstanceState)

程序的入口。

### §3.2 PageWidget类

**public class** PageWidget **extends** View

#### §3.2.1 摘要

##### ※属性

**private int mWidth** = 480

**private int mHeight** = 800

**private int mCornerX** = 0

**private int mCornerY**

**private** Path **mPath0**

**private** Path **mPath1**

Bitmap **mCurPageBitmap** = **null**

Bitmap **mNextPageBitmap** = **null**

PointF **mTouch** = **new** PointF()

PointF **mBezierStart1** = **new** PointF()

PointF **mBezierControl1** = **new** PointF()

PointF **mBeziervertex1** = **new** PointF()

PointF **mBezierEnd1** = **new** PointF()

PointF **mBezierStart2** = **new** PointF()

PointF **mBezierControl2** = **new** PointF()

PointF **mBeziervertex2** = **new** PointF()

PointF **mBezierEnd2** = **new** PointF()

**float mMiddleX**

**float mMiddleY**

**float mDegrees**

**float mTouchToCornerDis**

ColorMatrixColorFilter **mColorMatrixFilter**

Matrix **mMatrix**

**float**[] **mMatrixArray** = { 0, 0, 0, 0, 0, 0, 0, 0, 1.0f }

**boolean mIsRTandLB**

**float mMaxLength** = (**float**) Math.*hypot*(**mWidth**, **mHeight**)

**int**[] **mBackShadowColors**

**int**[] **mFrontShadowColors**

GradientDrawable **mBackShadowDrawableLR**

GradientDrawable **mBackShadowDrawableRL**

GradientDrawable **mFolderShadowDrawableLR**

GradientDrawable **mFolderShadowDrawableRL**

GradientDrawable **mFrontShadowDrawableHBT**

GradientDrawable **mFrontShadowDrawableHTB**

GradientDrawable **mFrontShadowDrawableVLR**

GradientDrawable **mFrontShadowDrawableVRL**

Paint **mPaint**

Scroller **mScroller**

##### ※方法

**public** PageWidget(Context context)

**public void** calcCornerXY(**float** x, **float** y)

**public boolean** doTouchEvent(MotionEvent event)

**public** PointF getCross(PointF P1, PointF P2, PointF P3, PointF P4)

**public void** setBitmaps(Bitmap bm1, Bitmap bm2)

**public void** drawCurrentPageShadow(Canvas canvas)

**public void** computeScroll()

**public void** abortAnimation()

**public boolean** canDragOver()

**public boolean** DragToRight()

**private void** calcPoints()

**private void** drawCurrentPageArea(Canvas canvas,

Bitmap bitmap, Path path)

**private void** drawNextPageAreaAndShadow(Canvas canvas,

Bitmap bitmap)

**private void** createDrawable()

**private void** drawCurrentBackArea(Canvas canvas,

Bitmap bitmap)

**private void** startAnimation(**int** delayMillis)

**protected void** onDraw(Canvas canvas)

#### §3.2.2 属性

##### ※mWidth

类型：**private int**

自定义组件PageWidget的宽度。

##### ※mHeight

类型：**private int**

自定义组件PageWidget的高度。

##### ※mCornerX

类型：**private int**

手指拖拽点对应的页角的x坐标。

注：一个页面总计4个页脚，获取其中之一的x坐标。

如下图手指触摸点所对应的页角应为右下角的那个角。



##### ※mCornerY

类型：**private int**

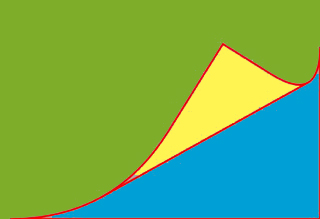
手指拖拽点对应的页角的y坐标。

注：一个页面总计4个页脚，获取其中之一的y坐标。

##### ※mPath0

类型：**private** Path

手指滑动翻页时，底下的一页显示出来的部分页面和上面的一页翻起后显示出来的的背面所组成的区域的边界（即下图蓝色和黄色所组成区域的边界）。



##### ※mPath1

类型：**private** Path

路径。

##### ※mCurPageBitmap

类型：Bitmap

当前页（即处于上面的那页）的位图。

##### ※mNextPageBitmap

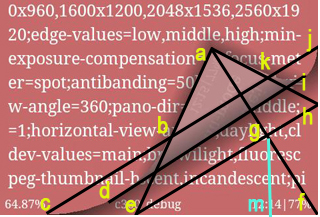
类型：Bitmap

下一页（即处于底下的那页）的位图。

##### ※mTouch

类型：PointF

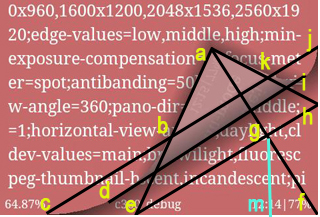
手指在组件上的拖拽点（即下图a点）。



##### ※mBezierStart1

类型：PointF

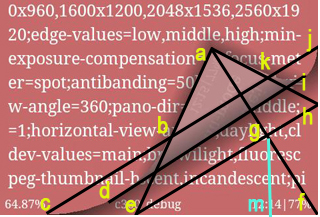
第一条贝塞尔曲线的起始点（即下图c点）。



##### ※mBezierControl1

类型：PointF

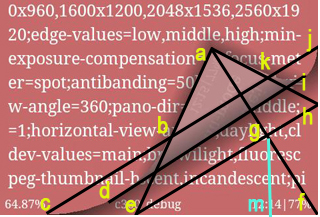
第一条贝塞尔曲线的控制点（即下图e点）。



##### ※mBeziervertex1

类型：PointF

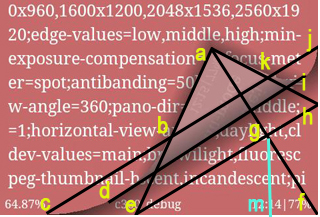
第一条贝塞尔曲线的顶点（即下图d点）。



##### ※mBezierEnd1

类型：PointF

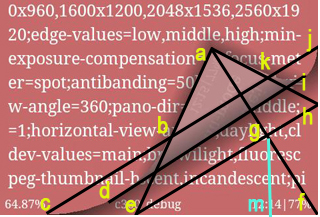
第一条贝塞尔曲线的终点（即下图b点）。



##### ※mBezierStart2

类型：PointF

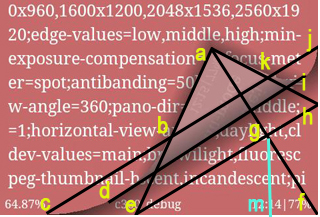
第二条贝塞尔曲线的起始点（即下图j点）。



##### ※mBezierControl2

类型：PointF

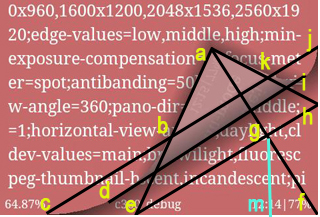
第二条贝塞尔曲线的控制点（即下图h点）。



##### ※mBeziervertex2

类型：PointF

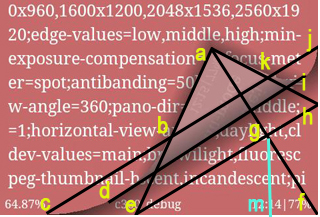
第二条贝塞尔曲线的顶点（即下图i点）。



##### ※mBezierEnd2

类型：PointF

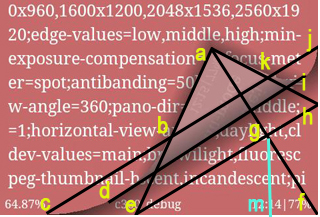
第二条贝塞尔曲线的终点（即下图k点）。



##### ※mMiddleX

类型：float

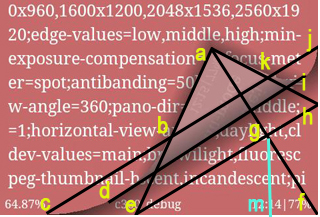
手指在组件上的拖拽点和手指拖拽点对应的页角的中点（即下图g点）的横坐标。**mMiddleX** = (**mTouch**.**x** + **mCornerX**) / 2;（在构造函数中初始化）。



##### ※mMiddleY

类型：float

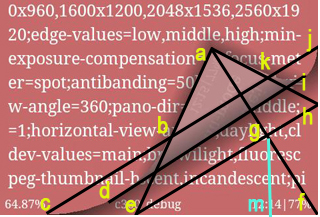
手指在组件上的拖拽点和手指拖拽点对应的页角的中点（即下图g点）的纵坐标。**mMiddleY** = (**mTouch**.**y** + **mCornerY**) / 2;（在构造函数中初始化）。



##### ※mDegrees

类型：float

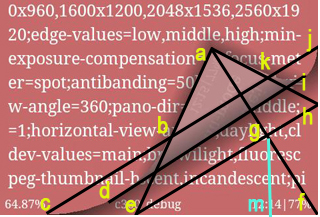
∠hef的度数（如下图）。



##### ※mTouchToCornerDis

类型：float

手指在组件上的拖拽点和手指拖拽点对应的页角之间的距离（即下图af）。



##### ※mColorMatrixFilter

类型：ColorMatrixColorFilter

通过4x5颜色矩阵转换颜色的滤色器。该滤镜可用于更改像素的饱和度，从YUV转换为RGB等。设置画笔的颜色。

##### ※mMatrix

类型：Matrix

矩阵对象。用于颜色的设置。

##### ※mMatrixArray

类型：**float**[]

浮点数组。用于设置上面的矩阵。

##### ※mIsRTandLB

类型：**boolean**

判断手指拖拽点对应的页角是否属于右上或者左下（是右上或者左下时值为true；不是右上或者左下时值为false）。

##### ※mMaxLength

类型：**float**

组件对角线的长度。

**mMaxLength** = (**float**) Math.*hypot*(**mWidth**, **mHeight**);

注：Math.*hypot方法返回所有参数的平方和的开方。*

##### ※mBackShadowColors

类型：**int**[]

背后阴影的颜色渐变范围。长度为2，从第一个元素代表的颜色渐变到第二个原色代表的颜色。

##### ※mFrontShadowColors

类型：**int**[]

前面阴影的颜色渐变范围。长度为2，从第一个元素代表的颜色渐变到第二个原色代表的颜色。

##### ※mBackShadowDrawableLR

类型：GradientDrawable

底下那页上从左到右的阴影渐变。

##### ※mBackShadowDrawableRL

类型：GradientDrawable

底下那页上从右到左的阴影渐变。

##### ※mFolderShadowDrawableLR

类型：GradientDrawable

折叠翻起的那页背面从左到右的阴影渐变。

##### ※mFolderShadowDrawableRL

类型：GradientDrawable

折叠翻起的那页背面从右到左的阴影渐变。

##### ※mFrontShadowDrawableHBT

类型：GradientDrawable

折叠翻起的那页前面水平从下到上的阴影渐变。

##### ※mFrontShadowDrawableHTB

类型：GradientDrawable

折叠翻起的那页前面水平从上到下的阴影渐变。

##### ※mFrontShadowDrawableVLR

类型：GradientDrawable

折叠翻起的那页前面垂直从左到右的阴影渐变。

##### ※mFrontShadowDrawableVRL

类型：GradientDrawable

折叠翻起的那页前面垂直从右到左的阴影渐变。

##### ※mPaint

类型：Paint

充当画笔，可以设置绘制时的笔触、颜色、样式等。

##### ※mScroller

类型：Scroller

响应滑动手势，实现翻页的动画。

#### §3.2.3 方法

##### ※PageWidget

**public** PageWidget(Context context)

参数：Context 上下文环境

构造方法。

*//父类构造方法*

**super**(context);

*//初始化路径*

**mPath0** = **new** Path();  
**mPath1** = **new** Path();

*//初始化GradientDrawable对象并给他们设置颜色，样式与渐变方式*

createDrawable();

*/\**

*初始化画笔*

*Paint.Style.FILL 只绘制图形内容  
\*/*

**mPaint** = **new** Paint();  
**mPaint**.setStyle(Paint.Style.***FILL***);

*//初始化具有相应效果的颜色矩阵，并用其初始化颜色过滤器*

ColorMatrix cm = **new** ColorMatrix();  
**float** array[] = { 0.55f, 0, 0, 0, 80.0f, 0, 0.55f, 0, 0, 80.0f, 0, 0,  
 0.55f, 0, 80.0f, 0, 0, 0, 0.2f, 0 };  
cm.set(array);  
**mColorMatrixFilter** = **new** ColorMatrixColorFilter(cm);

*//初始化矩阵*

**mMatrix** = **new** Matrix();

*//初始化数组以便后续给矩阵赋值*

float[] mMatrixArray = { 0, 0, 0, 0, 0, 0, 0, 0, 1.0f };

*//初始化滑动动作*

**mScroller** = **new** Scroller(getContext());  
*//初始化手指的触摸点*  
**mTouch**.**x** = 0.01f; *// 不让x,y为0,否则在点计算时会有问题***mTouch**.**y** = 0.01f;

##### ※calcCornerXY

**public void** calcCornerXY(**float** x, **float** y)

参数：**float 触摸点的**x坐标, **float 触摸点的**y坐标

作用：由传入的参数（x, y）（表示点的坐标）初始化触摸点所处的页角的坐标（mCornerX， mCornerY），并判断是否处在右上或者左下（即初始化mIsRTandLB）。

注：关于[安卓坐标系或者视图坐标系](#_安卓坐标系与视图坐标系)详见其他要点。

if (x <= **mWidth** / 2)  
 **mCornerX** = 0; *//页面的四个角*else  
 **mCornerX** = **mWidth**;  
if (y <= **mHeight** / 2)  
 **mCornerY** = 0;  
else  
 **mCornerY** = **mHeight**;  
if ((**mCornerX** == 0 && **mCornerY** == **mHeight**)  
 || (**mCornerX** == **mWidth** && **mCornerY** == 0))  
 **mIsRTandLB** = **true**; *//属于右上左下*else  
 **mIsRTandLB** = **false**; *//不属于右上左下*

##### ※doTouchEvent

**public boolean** doTouchEvent(MotionEvent event)

参数：MotionEvent 手指的动作

返回值：boolean

作用：手指在组件上落下滑动抬起时所出发的具体动作（事件）。

*//手指在组件上移动时实时获取触摸点的坐标并刷新重绘界面*

if (event.getAction() == MotionEvent.***ACTION\_MOVE***) {

*//event.getX()获取触摸点的横坐标，Y类似*   
 **mTouch.x** = event.getX();  
 **mTouch.y** = event.getY();  
 *//实现界面刷新重绘界面* **this**.postInvalidate();  
}

*//手指在组件上按下时实时获取触摸点的坐标*  
 if (event.getAction() == MotionEvent.***ACTION\_DOWN***) {

**mTouch.x** = event.getX();  
 **mTouch.y** = event.getY();  
}

*//手指在组件上抬起时判断翻起的那页是否能够翻过去，*

*//如果可以就开始翻页动画，如果不可以该页顺着翻起的轨迹落下，*

*//最后实时刷新重绘界面*  
 if (event.getAction() == MotionEvent.***ACTION\_UP***) {  
 if (canDragOver()) {

*//调用翻页动画*

startAnimation(1200);  
 } else {

*//将触摸点的坐标还原至相应的页脚处*  
 **mTouch.x** = **mCornerX** - 0.09f;  
 **mTouch.y** = **mCornerY** - 0.09f;  
 }

*//实现界面刷新重绘界面*  
 this.postInvalidate();  
 }  
 *// 总是返回true* return **true**;

##### ※getCross

**public** PointF getCross(PointF P1, PointF P2, PointF P3, PointF P4)

参数：PointF 点P1 ，PointF 点P2 ，PointF 点P3 ，PointF 点P4

返回值：PointF 交点

作用：求解直线P1P2和直线P3P4的交点的坐标。

PointF CrossP = **new** PointF();  
*// 二元函数通式： y=ax+b*

*//两直线交点的求法，此处不做过多说明*

**float** a1 = (P2.**y** - P1.**y**) / (P2.**x** - P1.**x**);  
**float** b1 = ((P1.**x** \* P2.**y**) - (P2.**x** \* P1.**y**)) / (P1.**x** - P2.**x**);  
**float** a2 = (P4.**y** - P3.**y**) / (P4.**x** - P3.**x**);  
**float** b2 = ((P3.**x** \* P4.**y**) - (P4.**x** \* P3.**y**)) / (P3.**x** - P4.**x**);  
CrossP.**x** = (b2 - b1) / (a1 - a2);  
CrossP.**y** = a1 \* CrossP.**x** + b1;  
**return** CrossP;

##### ※setBitmaps

**public void** setBitmaps(Bitmap bm1, Bitmap bm2)

参数：Bitmap 当前页的位图 ，Bitmap 下一页的位图

作用：设置当前上面一页的位图和底下一面的位图（位图可以看成画架）。

*// 直接用参数赋值（set方法）*

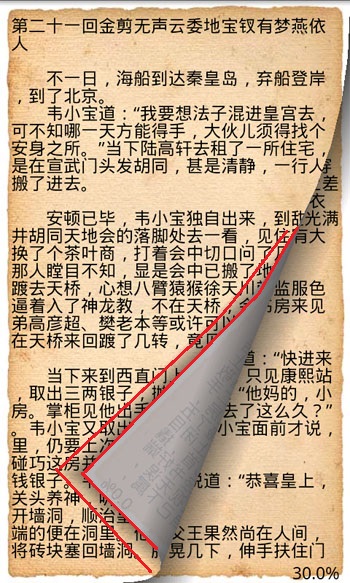
**mCurPageBitmap** = bm1;  
**mNextPageBitmap** = bm2;

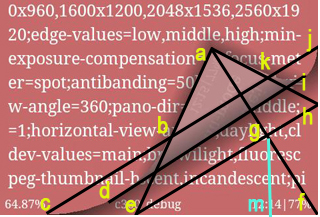
##### ※drawCurrentPageShadow

**public void** drawCurrentPageShadow(Canvas canvas)

参数：Canvas 画布

作用：绘制当前翻起的那页上面的阴影（即下图中红色圈出部分）。





*//∠aef的度数（如上图）*

**double** degree;

*//计算∠aef的度数即degree*  
 **if** (**mIsRTandLB**) {  
 degree = Math.***PI*** / 4  
 - Math.*atan2*(**mBezierControl1**.**y** - **mTouch**.**y**, **mTouch**.**x** - **mBezierControl1**.**x**);  
 } **else** {  
 *//45°减去一个角度* degree = Math.***PI*** / 4  
 - Math.*atan2*(**mTouch**.**y** - **mBezierControl1**.**y**, **mTouch**.**x** - **mBezierControl1**.**x**);  
 }  
 *// 翻起页阴影顶点与touch点的距离* **double** d1 = (**float**) 25 \* 1.414 \* Math.*cos*(degree);  
 **double** d2 = (**float**) 25 \* 1.414 \* Math.*sin*(degree);

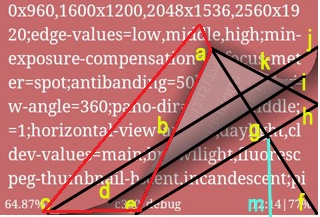
*// 翻起页阴影顶点的横坐标*  
 **float** x = (**float**) (**mTouch**.**x** + d1);

*// 翻起页阴影顶点的纵坐标*  
 **float** y;

*// 计算翻起页阴影顶点的纵坐标*  
 **if** (**mIsRTandLB**) {  
 y = (**float**) (**mTouch**.**y** + d2);  
 } **else** {  
 y = (**float**) (**mTouch**.**y** - d2);  
 }  
*// 清除路径中的所有直线和曲线，使其为空。这不会更改填充类型设置。* **mPath1**.reset();  
*// 将下一个轮廓的起点设置为点（x，y）。* **mPath1**.moveTo(x, y);  
*//从最后一个点到指定点（x，y）添加一条线。如果没有为此轮廓调用moveTo（），*

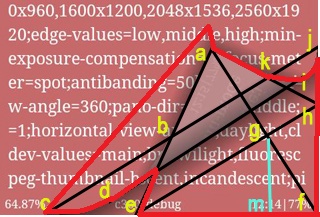
*// 则第一个点将自动设置为（0,0）* **mPath1**.lineTo(**mTouch**.**x**, **mTouch**.**y**);  
 **mPath1**.lineTo(**mBezierControl1**.**x**, **mBezierControl1**.**y**);  
 **mPath1**.lineTo(**mBezierStart1**.**x**, **mBezierStart1**.**y**);  
 **mPath1**.close();

*// mPath1的路径如下红色线标出所示*

**

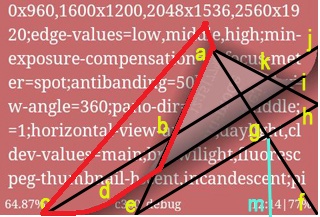
*// 旋转变换角度*  
 **float** rotateDegrees;  
 canvas.save();  
*// 使用指定的路径修改当前剪辑。  
// Region.Op.XOR:是全集形状减去交集形状之后的部分*

*// 中间桥梁（结果如下图）* canvas.clipPath(**mPath0**, Region.Op.***XOR***);

**

*//Region.Op.INTERSECT:A和B交集的形状*

*//求出上方一页上的会出现阴影的区域（结果如下图）* canvas.clipPath(**mPath1**, Region.Op.***INTERSECT***);

**

**int** leftx;  
 **int** rightx;

*//具有渐变的Drawable*  
 GradientDrawable mCurrentPageShadow;  
 **if** (**mIsRTandLB**) {  
 leftx = (**int**) (**mBezierControl1**.**x**);  
 rightx = (**int**) **mBezierControl1**.**x** + 25;  
 *//垂直左到右* mCurrentPageShadow = **mFrontShadowDrawableVLR**;  
 } **else** {  
 leftx = (**int**) (**mBezierControl1**.**x** - 25);  
 rightx = (**int**) **mBezierControl1**.**x** + 1;  
 mCurrentPageShadow = **mFrontShadowDrawableVRL**;  
 }

*// 计算旋转角度。*  
 rotateDegrees = (**float**) Math.*toDegrees*(Math.*atan2*(**mTouch**.**x** - **mBezierControl1**.**x**, **mBezierControl1**.**y** - **mTouch**.**y**));

*// 图形旋转变换。*  
 canvas.rotate(rotateDegrees, **mBezierControl1**.**x**, **mBezierControl1**.**y**);  
*// 为Drawable指定边界矩形。* mCurrentPageShadow.setBounds(leftx,  
 (**int**) (**mBezierControl1**.**y** - **mMaxLength**), rightx,  
 (**int**) (**mBezierControl1**.**y**));  
*// 根据可选效果绘制边界（通过setBounds设置），例如alpha（通过setAlpha设置）和滤色器（通过setColorFilter设置）。* mCurrentPageShadow.draw(canvas);  
 canvas.restore();  
 *//同理，处理另一边（处理过程和上面一样）* **mPath1**.reset();  
 **mPath1**.moveTo(x, y);  
 **mPath1**.lineTo(**mTouch**.**x**, **mTouch**.**y**);  
 **mPath1**.lineTo(**mBezierControl2**.**x**, **mBezierControl2**.**y**);  
 **mPath1**.lineTo(**mBezierStart2**.**x**, **mBezierStart2**.**y**);  
 **mPath1**.close();  
 canvas.save();  
 canvas.clipPath(**mPath0**, Region.Op.***XOR***);  
 canvas.clipPath(**mPath1**, Region.Op.***INTERSECT***);  
 **if** (**mIsRTandLB**) {  
 leftx = (**int**) (**mBezierControl2**.**y**);  
 rightx = (**int**) (**mBezierControl2**.**y** + 25);  
 *//垂直上到下* mCurrentPageShadow = **mFrontShadowDrawableHTB**;  
 } **else** {  
 leftx = (**int**) (**mBezierControl2**.**y** - 25);  
 rightx = (**int**) (**mBezierControl2**.**y** + 1);  
 mCurrentPageShadow = **mFrontShadowDrawableHBT**;  
 }  
 rotateDegrees = (**float**) Math.*toDegrees*(Math.*atan2*(**mBezierControl2**.**y** - **mTouch**.**y**, **mBezierControl2**.**x** - **mTouch**.**x**));  
 canvas.rotate(rotateDegrees, **mBezierControl2**.**x**, **mBezierControl2**.**y**);  
 **float** temp;  
 **if** (**mBezierControl2**.**y** < 0)  
 temp = **mBezierControl2**.**y** - **mHeight**;  
 **else** temp = **mBezierControl2**.**y**;  
 **int** hmg = (**int**) Math.*hypot*(**mBezierControl2**.**x**, temp);  
 **if** (hmg > **mMaxLength**)  
 mCurrentPageShadow  
 .setBounds((**int**) (**mBezierControl2**.**x** - 25) - hmg, leftx,  
 (**int**) (**mBezierControl2**.**x** + **mMaxLength**) - hmg,  
 rightx);  
 **else** mCurrentPageShadow.setBounds(  
 (**int**) (**mBezierControl2**.**x** - **mMaxLength**), leftx,  
 (**int**) (**mBezierControl2**.**x**), rightx);mCurrentPageShadow.draw(canvas);  
 canvas.restore();

##### ※computeScroll

**public void** computeScroll()

作用：该方法为父类View类中computeScroll()方法的重写。

computeScroll不是来让View滑动的。

主要功能是计算拖动的位移量、更新背景、设置要显示的屏幕，计算ViewGroup如何滑动。computeScroll是通过draw来调用的。

重写computeScroll()的原因：调用startScroll()是不会有滚动效果的，只有在computeScroll()获取滚动情况，做出滚动的响应，computeScroll在父控件执行drawChild时，会调用这个方法。

何时调用computeScroll：因为computeScroll调用Scroller，所以只要computeScroll调用连续，Scroller也会连续，实质上computeScroll的连续性又invalidate方法控制，scrollTo,scrollBy都会调用invalidate，而invalidate回去触发draw,从而computeScroll被连续调用，综上，Scroller会被连续调用，除非invalidate停止调用。

computeScroll参考Scroller影响scrollTo,scrollBy，为了不重复影响scrollTo,scrollBy，那么Scroller必须终止计算currX，currY。要知道计算有没有终止，需要通过mScroller.computeScrollOffset()。

*//父类构造方法*

**super**.computeScroll();  
*// 计算有没有终止，需要通过mScroller.computeScrollOffset()  
// 判断移动过程是否完成* **if** (**mScroller**.computeScrollOffset()) {  
 **float** x = **mScroller**.getCurrX();  
 **float** y = **mScroller**.getCurrY();  
 **mTouch**.**x** = x;  
 **mTouch**.**y** = y;  
 postInvalidate();  
 }

##### ※abortAnimation

**public void** abortAnimation()

作用：如果滚动没有结束，调用Scroller类的abortAnimation()，终止正在进行的动画，终止动画会导致滚动条移动到最终的（x, y）的位置。

此函数用于自动翻到下一页或者落回原处时，若手指点击某个部位，则将动画终止并将翻起的页角直接移动到手指点的部位。

**if** (!**mScroller**.isFinished()) {  
 **mScroller**.abortAnimation();  
}

##### ※canDragOver

**public boolean** canDragOver()

返回值：boolean

作用：判断能否自动翻到下一页或者翻到上一页。

如果手指触摸点到相应页角的距离大于组件宽度的十分之一时，可以自动翻页。

**if** (**mTouchToCornerDis** > **mWidth** / 10)  
 **return true**;  
**return false**;

##### ※DragToRight

**public boolean** DragToRight()

返回值：boolean

作用：是否是从左边翻到右边（是为true，否为false）。

如果手指触摸点所对应的页角是大于0的就说明是从右边翻到左边，所以返回false（如下图手指触摸点所对应的页角应为右下角的那个角）；反之亦然。

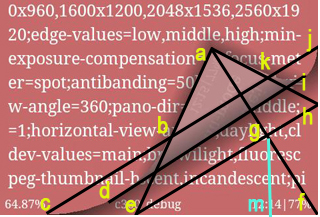


**if** (**mCornerX** > 0)  
 **return false**;  
**return true**;

##### ※calcPoints

**private void** calcPoints()

作用：计算一些在画图中需要用到的点的坐标。



*//g点坐标* **mMiddleX** = (**mTouch**.**x** + **mCornerX**) / 2;  
 **mMiddleY** = (**mTouch**.**y** + **mCornerY**) / 2;  
 *//e点坐标* **mBezierControl1**.**x** = **mMiddleX** - (**mCornerY** - **mMiddleY**)  
 \* (**mCornerY** - **mMiddleY**) / (**mCornerX** - **mMiddleX**);  
 **mBezierControl1**.**y** = **mCornerY**;  
 *//h点坐标* **mBezierControl2**.**x** = **mCornerX**;  
 **mBezierControl2**.**y** = **mMiddleY** - (**mCornerX** - **mMiddleX**)  
 \* (**mCornerX** - **mMiddleX**) / (**mCornerY** - **mMiddleY**);  
  *//c点坐标* **mBezierStart1**.**x** = **mBezierControl1**.**x** - (**mCornerX** - **mBezierControl1**.**x**)  
 / 2;  
 **mBezierStart1**.**y** = **mCornerY**;  
 *// 当mBezierStart1.x < 0或者mBezierStart1.x > 480时  
 // 如果继续翻页，会出现BUG故在此限制*

*// 所以当mBezierStart1.x < 0或者mBezierStart1.x > 480时，经过变换使0 <= mBezierStart1.x <= 480，同时其余的点也应该经过相应的变换以保持相对不变性。* **if** (**mTouch**.**x** > 0 && **mTouch**.**x** < **mWidth**) {  
 **if** (**mBezierStart1**.**x** < 0 || **mBezierStart1**.**x** > **mWidth**) {  
 **if** (**mBezierStart1**.**x** < 0)  
 **mBezierStart1**.**x** = **mWidth** - **mBezierStart1**.**x**;  
 **float** f1 = Math.*abs*(**mCornerX** - **mTouch**.**x**);  
 **float** f2 = **mWidth** \* f1 / **mBezierStart1**.**x**;  
 **mTouch**.**x** = Math.*abs*(**mCornerX** - f2);  
 **float** f3 = Math.*abs*(**mCornerX** - **mTouch**.**x**)  
 \* Math.*abs*(**mCornerY** - **mTouch**.**y**) / f1;  
 **mTouch**.**y** = Math.*abs*(**mCornerY** - f3);  
 **mMiddleX** = (**mTouch**.**x** + **mCornerX**) / 2;  
 **mMiddleY** = (**mTouch**.**y** + **mCornerY**) / 2;  
 **mBezierControl1**.**x** = **mMiddleX** - (**mCornerY** - **mMiddleY**)  
 \* (**mCornerY** - **mMiddleY**) / (**mCornerX** - **mMiddleX**);  
 **mBezierControl1**.**y** = **mCornerY**;  
 **mBezierControl2**.**x** = **mCornerX**;  
 **mBezierControl2**.**y** = **mMiddleY** - (**mCornerX** - **mMiddleX**)  
 \* (**mCornerX** - **mMiddleX**) / (**mCornerY** - **mMiddleY**);**mBezierStart1**.**x** = **mBezierControl1**.**x** - (**mCornerX** - **mBezierControl1**.**x**) / 2;  
 }  
 }  
 *//j点坐标* **mBezierStart2**.**x** = **mCornerX**;  
 **mBezierStart2**.**y** = **mBezierControl2**.**y** - (**mCornerY** - **mBezierControl2**.**y**)  
 / 2;  
*// Math.hypot方法返回所有参数的平方和的平方根。  
 //af的长度* **mTouchToCornerDis** = (**float**) Math.*hypot*((**mTouch**.**x** - **mCornerX**),  
 (**mTouch**.**y** - **mCornerY**));  
 *//b点坐标* **mBezierEnd1** = getCross(**mTouch**, **mBezierControl1**, **mBezierStart1**,  
 **mBezierStart2**);  
 *//k点坐标* **mBezierEnd2** = getCross(**mTouch**, **mBezierControl2**, **mBezierStart1**,  
 **mBezierStart2**); *//d点坐标* **mBeziervertex1**.**x** = (**mBezierStart1**.**x** + 2 \* **mBezierControl1**.**x** + **mBezierEnd1**.**x**) / 4;  
 **mBeziervertex1**.**y** = (2 \* **mBezierControl1**.**y** + **mBezierStart1**.**y** + **mBezierEnd1**.**y**) / 4;  
 *//i点坐标* **mBeziervertex2**.**x** = (**mBezierStart2**.**x** + 2 \* **mBezierControl2**.**x** + **mBezierEnd2**.**x**) / 4;  
 **mBeziervertex2**.**y** = (2 \* **mBezierControl2**.**y** + **mBezierStart2**.**y** + **mBezierEnd2**.**y**) / 4;

##### ※drawCurrentPageArea

**private void** drawCurrentPageArea(Canvas canvas, Bitmap bitmap, Path path)

参数：Canvas 画布, Bitmap 位图, Path 路径

作用：绘制出当前（即处于上方）的那一页（如下图）。

*// 重置mPath0*

**mPath0**.reset();

*// 设置路径起点到图中e点*  
 **mPath0**.moveTo(**mBezierStart1**.**x**, **mBezierStart1**.**y**);  
*// 贝赛尔曲线的实现  
// 第一个参数：控制点的x坐标  
// 第二个参数：控制点的y坐标  
// 第三个参数：结束点的x坐标  
// 第四个参数：结束点的y坐标* **mPath0**.quadTo(**mBezierControl1**.**x**, **mBezierControl1**.**y**, **mBezierEnd1**.**x**,  
 **mBezierEnd1**.**y**);  
 *//画直线* **mPath0**.lineTo(**mTouch**.**x**, **mTouch**.**y**);  
 **mPath0**.lineTo(**mBezierEnd2**.**x**, **mBezierEnd2**.**y**);  
 **mPath0**.quadTo(**mBezierControl2**.**x**, **mBezierControl2**.**y**, **mBezierStart2**.**x**,  
 **mBezierStart2**.**y**);  
 **mPath0**.lineTo(**mCornerX**, **mCornerY**);  
*// 如果连接Path起点和终点能形成一个闭合图形，则会将起点和终点连接起来形成一个闭合图形（效果如下图）* **mPath0**.close();

**

canvas.save();

*// 与传入的参数进行剪辑*  
 canvas.clipPath(path, Region.Op.***XOR***)  
*// 把bitmap显示到left, top所指定的左上角位置.  
// 第一个参数为要绘制的bitmap对象，  
// 第二个参数为图片左上角的x坐标值，  
// 第三个参数为图片左上角的y坐标的值，  
// 第三个参数为Paint对象。* canvas.drawBitmap(bitmap, 0, 0, **null**);  
 canvas.restore();

##### ※drawNextPageAreaAndShadow

**private void** drawNextPageAreaAndShadow(Canvas canvas, Bitmap bitmap)

参数：Canvas 画布, Bitmap 位图

作用：绘制底下那个页面上显示出来的区域以及该区域上的阴影。

*//下一页的区域* **mPath1**.reset();  
 **mPath1**.moveTo(**mBezierStart1**.**x**, **mBezierStart1**.**y**);  
 **mPath1**.lineTo(**mBeziervertex1**.**x**, **mBeziervertex1**.**y**);  
 **mPath1**.lineTo(**mBeziervertex2**.**x**, **mBeziervertex2**.**y**);  
 **mPath1**.lineTo(**mBezierStart2**.**x**, **mBezierStart2**.**y**);  
 **mPath1**.lineTo(**mCornerX**, **mCornerY**);  
 **mPath1**.close();  
 *//效果如下所示*



*// toDegrees() 方法用于将参数转化为角度。  
// Math.atan2函数有两个参数x，y。该函数返回的值也是一个弧度值。  
// 它代表的是坐标(0,0)指向坐标(x,y)的向量方向和x轴坐标的角度值。  
// double angle = Math.atan2(x,y);  
 //∠hef的度数* **mDegrees** = (**float**) Math.*toDegrees*(Math.*atan2*(**mBezierControl1**.**x** - **mCornerX**, **mBezierControl2**.**y** - **mCornerY**));  
 **int** leftx; *//阴影的左端点* **int** rightx; *//阴影的右端点* GradientDrawable mBackShadowDrawable;  
 **if** (**mIsRTandLB**) { *//如果左下或者右上时* leftx = (**int**) (**mBezierStart1**.**x**); *//c点横坐标* rightx = (**int**) (**mBezierStart1**.**x** + **mTouchToCornerDis** / 4);  
 *//左上到右下* mBackShadowDrawable = **mBackShadowDrawableLR**;  
 } **else** {  
 leftx = (**int**) (**mBezierStart1**.**x** - **mTouchToCornerDis** / 4);  
 rightx = (**int**) **mBezierStart1**.**x**;  
 *//右上到左下* mBackShadowDrawable = **mBackShadowDrawableRL**;  
 }  
 canvas.save();

*//绘制阴影部分*

canvas.clipPath(**mPath0**);  
 canvas.clipPath(**mPath1**, Region.Op.***INTERSECT***);  
 canvas.drawBitmap(bitmap, 0, 0, **null**);  
 *//画布的旋转  
 //第一个参数为旋转角度  
 //第二个参数为基准点的x  
 //第三个参数为基准点的y* canvas.rotate(**mDegrees**, **mBezierStart1**.**x**, **mBezierStart1**.**y**);  
*// 为Drawable指定边界矩形。这是可绘制对象在调用其draw（）方法时绘制的位置。  
// （int left,  
// int top,  
// int right,  
// int bottom）* mBackShadowDrawable.setBounds(leftx, (**int**) **mBezierStart1**.**y**, rightx,  
 (**int**) (**mMaxLength** + **mBezierStart1**.**y**));  
*// 根据可选效果绘制边界（通过setBounds设置），例如alpha（通过setAlpha设置）和滤色器（通过setColorFilter设置）* mBackShadowDrawable.draw(canvas);  
 canvas.restore();

##### ※createDrawable

**private void** createDrawable()

作用：创建阴影的GradientDrawable（具有颜色渐变的Drawable）。

*// 颜色矩阵，从第一个颜色渐变到第二个颜色*

**int**[] color = { 0x333333, 0xb0333333 };  
*// 给定方向和渐变的颜色数组，创建一个新的渐变可绘制对象。  
// 从右向左绘制渐变* **mFolderShadowDrawableRL** = **new** GradientDrawable(  
 GradientDrawable.Orientation.***RIGHT\_LEFT***, color);  
*// 设置此可绘制对象使用的渐变类型* **mFolderShadowDrawableRL**.setGradientType(GradientDrawable.***LINEAR\_GRADIENT***);  
*// 从左到右绘制渐变* **mFolderShadowDrawableLR** = **new** GradientDrawable(  
 GradientDrawable.Orientation.***LEFT\_RIGHT***, color);  
*// 设置此可绘制对象使用的渐变类型* **mFolderShadowDrawableLR**.setGradientType(GradientDrawable.***LINEAR\_GRADIENT***);

*// 颜色矩阵，从第一个颜色渐变到第二个颜色（与上面的差不多，只不过这个是属性）*  
 **mBackShadowColors** = **new int**[] { 0xff111111, 0x111111 };  
 **mBackShadowDrawableRL** = **new** GradientDrawable(  
 GradientDrawable.Orientation.***RIGHT\_LEFT***, **mBackShadowColors**);  
 **mBackShadowDrawableRL**.setGradientType(GradientDrawable.***LINEAR\_GRADIENT***);  
 **mBackShadowDrawableLR** = **new** GradientDrawable(  
 GradientDrawable.Orientation.***LEFT\_RIGHT***, **mBackShadowColors**);  
 **mBackShadowDrawableLR**.setGradientType(GradientDrawable.***LINEAR\_GRADIENT***);  
 **mFrontShadowColors** = **new int**[] { 0x80111111, 0x111111 };  
 **mFrontShadowDrawableVLR** = **new** GradientDrawable(  
 GradientDrawable.Orientation.***LEFT\_RIGHT***, **mFrontShadowColors**);  
**mFrontShadowDrawableVLR**.setGradientType(GradientDrawable.***LINEAR\_GRADIENT***);  
 **mFrontShadowDrawableVRL** = **new** GradientDrawable(  
 GradientDrawable.Orientation.***RIGHT\_LEFT***, **mFrontShadowColors**); **mFrontShadowDrawableVRL**.setGradientType(GradientDrawable.***LINEAR\_GRADIENT***);  
 **mFrontShadowDrawableHTB** = **new** GradientDrawable(  
 GradientDrawable.Orientation.***TOP\_BOTTOM***, **mFrontShadowColors**);  
**mFrontShadowDrawableHTB**.setGradientType(GradientDrawable.***LINEAR\_GRADIENT***);  
 **mFrontShadowDrawableHBT** = **new** GradientDrawable(  
 GradientDrawable.Orientation.***BOTTOM\_TOP***, **mFrontShadowColors**); **mFrontShadowDrawableHBT**.setGradientType(GradientDrawable.***LINEAR\_GRADIENT***);

##### ※drawCurrentBackArea

**private void** drawCurrentBackArea(Canvas canvas, Bitmap bitmap)

参数：Canvas 画布, Bitmap 位图

作用：绘制当前翻起页的背面(绘图过程以及路径的生成类似其他的)。

**int** i = (**int**) (**mBezierStart1**.**x** + **mBezierControl1**.**x**) / 2;  
 **float** f1 = Math.*abs*(i - **mBezierControl1**.**x**);  
 **int** i1 = (**int**) (**mBezierStart2**.**y** + **mBezierControl2**.**y**) / 2;  
 **float** f2 = Math.*abs*(i1 - **mBezierControl2**.**y**);  
 **float** f3 = Math.*min*(f1, f2);  
 **mPath1**.reset();  
 **mPath1**.moveTo(**mBeziervertex2**.**x**, **mBeziervertex2**.**y**);  
 **mPath1**.lineTo(**mBeziervertex1**.**x**, **mBeziervertex1**.**y**);  
 **mPath1**.lineTo(**mBezierEnd1**.**x**, **mBezierEnd1**.**y**);  
 **mPath1**.lineTo(**mTouch**.**x**, **mTouch**.**y**);  
 **mPath1**.lineTo(**mBezierEnd2**.**x**, **mBezierEnd2**.**y**);  
 **mPath1**.close();  
 GradientDrawable mFolderShadowDrawable;  
 **int** left;  
 **int** right;  
 **if** (**mIsRTandLB**) {  
 left = (**int**) (**mBezierStart1**.**x** - 1);  
 right = (**int**) (**mBezierStart1**.**x** + f3 + 1);  
 mFolderShadowDrawable = **mFolderShadowDrawableLR**;  
 } **else** {  
 left = (**int**) (**mBezierStart1**.**x** - f3 - 1);  
 right = (**int**) (**mBezierStart1**.**x** + 1);  
 mFolderShadowDrawable = **mFolderShadowDrawableRL**;  
 }  
 canvas.save();  
 canvas.clipPath(**mPath0**);  
 canvas.clipPath(**mPath1**, Region.Op.***INTERSECT***);  
 **mPaint**.setColorFilter(**mColorMatrixFilter**);  
 *//eh的长度* **float** dis = (**float**) Math.*hypot*(**mCornerX** - **mBezierControl1**.**x**,  
 **mBezierControl2**.**y** - **mCornerY**);  
 *//cos(∠hef)* **float** f8 = (**mCornerX** - **mBezierControl1**.**x**) / dis;  
 *//sin(∠hef)* **float** f9 = (**mBezierControl2**.**y** - **mCornerY**) / dis;  
 **mMatrixArray**[0] = 1 - 2 \* f9 \* f9;  
 **mMatrixArray**[1] = 2 \* f8 \* f9;  
 **mMatrixArray**[3] = **mMatrixArray**[1];  
 **mMatrixArray**[4] = 1 - 2 \* f8 \* f8;  
 **mMatrix**.reset();  
*// 将数组中的9个值复制到矩阵中。根据Matrix的实现，  
// 可以将它们转换为Matrix中的16.16整数，  
// 这样，随后对getValues（）的调用将不会产生完全相同的值。* **mMatrix**.setValues(**mMatrixArray**);  
*// 用指定的变换右乘。M'= M \* T（dx，dy）* **mMatrix**.preTranslate(-**mBezierControl1**.**x**, -**mBezierControl1**.**y**);  
*// 用指定的变换左乘。M'= T（dx，dy）\* M* **mMatrix**.postTranslate(**mBezierControl1**.**x**, **mBezierControl1**.**y**);  
 canvas.drawBitmap(bitmap, **mMatrix**, **mPaint**);  
**mPaint**.setColorFilter(**null**);  
 canvas.rotate(**mDegrees**, **mBezierStart1**.**x**, **mBezierStart1**.**y**);  
 mFolderShadowDrawable.setBounds(left, (**int**) **mBezierStart1**.**y**, right,  
 (**int**) (**mBezierStart1**.**y** + **mMaxLength**));  
 mFolderShadowDrawable.draw(canvas);  
 canvas.restore();

##### ※startAnimation

**private void** startAnimation(**int** delayMillis)

参数：**int 延迟的时间（秒）**

作用：开始翻页的动画（指手指翻起一页到一定程度松手就会自动翻页）。

**int** dx, dy;  
 *// dx 水平方向滑动的距离，负值会使滚动向左滚动  
 // dy 垂直方向滑动的距离，负值会使滚动向上滚动* **if** (**mCornerX** > 0) {  
 dx = -(**int**) (**mWidth** + **mTouch**.**x**);  
 } **else** {  
 dx = (**int**) (**mWidth** - **mTouch**.**x** + **mWidth**);  
 }  
 **if** (**mCornerY** > 0) {  
 dy = (**int**) (**mHeight** - **mTouch**.**y**);  
 } **else** {  
 dy = (**int**) (1 - **mTouch**.**y**); *// 防止mTouch.y最终变为0* }  
*// 通过提供起点，行进距离和滚动持续时间来开始滚动。  
// startX Int：开始的水平滚动偏移量（以像素为单位）。正数会将内容向左滚动。  
// startY Int：开始的垂直滚动偏移量（以像素为单位）。正数将使内容向上滚动。  
// dx Int：行进的水平距离。正数会将内容向左滚动。  
// dy Int：行进的垂直距离。正数将使内容向上滚动。  
// duration Int：滚动持续时间（以毫秒为单位）。* **mScroller**.startScroll((**int**) **mTouch**.**x**, (**int**) **mTouch**.**y**, dx, dy,  
 delayMillis);

##### ※onDraw

**protected void** onDraw(Canvas canvas)

参数：Canvas 画布

作用：该方法为父类View类中onDraw()方法的重写(onDraw是View的三大流程之一)。

onDraw()方法的调用时间：这个方法会在初始化后被调用一次,invaildate()（即刷新重绘）的时候也会被调用。

*// 使用srcover porterduff模式用指定的颜色填充整个画布的位图（仅限于当前剪辑）。  
// 在整个绘制区域统一涂上指定的颜色。* canvas.drawColor(0xFFAAAAAA);

*// 计算关键点*

calcPoints();  
*// 绘制当前页上（即上面一页显示出来的那一部分）的内容*

drawCurrentPageArea(canvas, **mCurPageBitmap**, **mPath0**);

*// 绘制底下那一页显示出来的部分和该部分上的阴影* drawNextPageAreaAndShadow(canvas, **mNextPageBitmap**);  
*// 绘制当前页（即上面一页显示出来的那一部分）上的阴影*  
 drawCurrentPageShadow(canvas);  
*// 绘制当前页背面上的阴影*  
 drawCurrentBackArea(canvas, **mCurPageBitmap**);

### §3.3 BookPageFactory类

**public class** BookPageFactory

#### §3.3.1 摘要

##### ※属性

**private** File **book\_file** = **null**

**private** MappedByteBuffer **m\_mbBuf** = **null**

**private int m\_mbBufLen** = 0

**private int m\_mbBufBegin** = 0

**private int m\_mbBufEnd** = 0

**private** String **m\_strCharsetName** = **"GBK"**

**private** Bitmap **m\_book\_bg** = **null**

**private int mWidth**

**private int mHeight**

**private** Vector<String> **m\_lines** = **new** Vector<String>()

**private int m\_fontSize** = 24

**private int m\_textColor** = Color.***BLACK***

**private int m\_backColor** = 0xffff9e85

**private int marginWidth** = 15

**private int marginHeight** = 20

**private int mLineCount**

**private float mVisibleHeight**

**private float mVisibleWidth**

**private boolean m\_isfirstPage**

**private boolean m\_islastPage**

**private** Paint **mPaint**

##### ※方法

**public** BookPageFactory(**int** w, **int** h)

**public void** openbook(String strFilePath)

**public void** nextPage()

**public void** onDraw(Canvas c)

**public void** setBgBitmap(Bitmap BG)

**public boolean** isfirstPage()

**public boolean** islastPage()

**protected byte**[] readParagraphBack(**int** nFromPos)

**protected byte**[] readParagraphForward(**int** nFromPos)

**protected** Vector<String> pageDown()

**protected void** pageUp()

**protected void** prePage()

#### §3.3.2 属性

##### ※book\_file

类型：**private** File

要打开读取的txt文件。

##### ※m\_mbBuf

类型：**private** MappedByteBuffer

文件缓冲区。基于MappedByteBuffer操作txt文件。将文件映射到内存中，可以高效地处理文件。

##### ※m\_mbBufLen

类型：**private int**

文件缓冲区的长度。

##### ※m\_mbBufBegin

类型：**private int**

文件缓冲区的开始下标。

##### ※m\_mbBufEnd

类型：**private int**

文件缓冲区的结束下标。

##### ※m\_strCharsetName

类型：**private** String

文件内容的字符编码方式。

##### ※m\_book\_bg

类型：**private** Bitmap

放置电子书的背景图片的位图。

##### ※mWidth

类型：**private int**

电子书每一页的宽度。

##### ※mHeight

类型：**private int**

电子书每一页的高度。

##### ※m\_lines

类型：**private** Vector<String>

一个元素是String字符串的数组。数组中元素从小到大分别为一页上从上到下的每行文字。

##### ※m\_fontSize

类型：**private int**

电子书的字号大小。

##### ※m\_textColor

类型：**private int**

电子书的字体颜色。

##### ※m\_backColor

类型：**private int**

电子书的背景颜色（假如没有背景图片的话显示背景颜色）。

##### ※marginWidth

类型：**private int**

电子书每一页上的文字区域的左右两端与整个电子书版本的左右边缘的距离（即左方右方边缘间距）。

##### ※marginHeight

类型：**private int**

电子书每一页上的文字区域的上下两端与整个电子书版本的上下边缘的距离（即上方下方边缘间距）。

##### ※mLineCount

类型：**private int**

电子书每一页上可以显示的文字的行数。

##### ※mVisibleHeight

类型：**private float**

电子书每一页上的文字区域的高度（即绘制文字的区域的高度）。

##### ※mVisibleWidth

类型：**private float**

电子书每一页上的文字区域的宽度（即绘制文字的区域的宽度）。

##### ※m\_isfirstPage

类型：**private boolean**

电子书是否处在第一页（在第一页时值为true，不在第一页时值为false）。

##### ※m\_islastPage

类型：**private boolean**

电子书是否处在最后一页（在最后一页时值为true，不在最后一页时值为false）。

##### ※mPaint

类型：Paint

充当画笔，可以设置绘制时的笔触、颜色、样式等。

#### §3.3.3 方法

##### ※BookPageFactory

**public** BookPageFactory(**int** w, **int** h)

参数：**int** 宽 ，**int** 高

构造函数。

**mWidth** = w; *//组件宽度*  
**mHeight** = h; *//组件宽度*

*//设置画笔的属性*  
**mPaint** = **new** Paint(Paint.***ANTI\_ALIAS\_FLAG***); *//抗锯齿***mPaint**.setTextAlign(Align.***LEFT***); *//左对齐***mPaint**.setTextSize(**m\_fontSize**); *//设置字号***mPaint**.setColor(**m\_textColor**); *//设置字体颜色***mVisibleWidth** = **mWidth** - **marginWidth** \* 2; *//绘制的宽***mVisibleHeight** = **mHeight** - **marginHeight** \* 2; *//绘制的高***mLineCount** = (**int**) (**mVisibleHeight** / **m\_fontSize**); *// 可显示的行数*

##### ※openbook

**public void** openbook(String strFilePath)

**throws** IOException

参数：String文件路径

作用：打开txt文件。读取文件到内存缓冲区中去。

**book\_file** = **new** File(strFilePath); *//给book\_file赋值*  
**long** lLen = **book\_file**.length(); *//返回由此抽象路径名表示的文件的长度***m\_mbBufLen** = (**int**) lLen; *//给长度m\_mbBufLen赋值*  
*// RandomAccessFile(book\_file, "r")：创建从中读取和向其中写入（可选）的随机访问文件流，该文件由 File 参数指定  
// getChannel()：返回与此文件关联的唯一 FileChannel 对象  
// map(FileChannel.MapMode.READ\_ONLY, 0, lLen)：将此通道的文件区域直接映射到内存中。  
// 参数：  
// mode - 根据是按只读、读取/写入或专用（写入时拷贝）来映射文件，分别为 FileChannel.MapMode 类中所定义的 READ\_ONLY、READ\_WRITE 或 PRIVATE 之一  
// position - 文件中的位置，映射区域从此位置开始；必须为非负数  
// size - 要映射的区域大小；必须为非负数且不大于 Integer.MAX\_VALUE  
// 返回MappedByteBuffer ：直接字节缓冲区，其内容是文件的内存映射区域。* **m\_mbBuf** = **new** RandomAccessFile(**book\_file**, **"r"**).getChannel().map(  
 FileChannel.MapMode.***READ\_ONLY***, 0, lLen);

##### ※nextPage

**public void** nextPage()

**throws** IOException

作用：判断是否到达最后一页（即书尾）给m\_islastPage赋值。若没有到最后一页就将下一页的内容放到m\_lines中，并将当前页的末尾缓冲区指针赋给下一页的起始缓冲区指针，准备进行下一页的读取。

*// 到达末尾*

**if** (**m\_mbBufEnd** >= **m\_mbBufLen**) {  
 **m\_islastPage**=**true**;  
 **return**;  
}**else**

*// 未到达末尾* **m\_islastPage**=**false**;  
 **m\_lines**.clear();

*// 更新缓冲区指针*  
 **m\_mbBufBegin** = **m\_mbBufEnd**;

*// 更新m\_lines*  
 **m\_lines** = pageDown();

##### ※onDraw

**public void** onDraw(Canvas c)

参数：Canvas画布。

作用：设置背景并将一页上的文字内容绘制出来，同时也要计算已读百分比并绘制出来。

**if** (**m\_lines**.size() == 0)  
 **m\_lines** = pageDown();  
 **if** (**m\_lines**.size() > 0) {  
 **if** (**m\_book\_bg** == **null**)

*// 背景图片为空，就设置背景颜色*  
 c.drawColor(**m\_backColor**);  
 **else**

*// 背景图片不为空，设置背景图片*c.drawBitmap(**m\_book\_bg**, 0, 0, **null**);  
 **int** y = **marginHeight**;  
 **for** (String strLine : **m\_lines**) {

*// 一行一行的绘制，y每次都更新，y为每行起始的y坐标*  
 y += **m\_fontSize**;  
*// text:要绘制的文字即此处的strLine  
// x：绘制原点x坐标即此处的marginWidth  
// y：绘制原点y坐标即此处的y  
// paint:用来做画的画笔即此处的mPaint* c.drawText(strLine, **marginWidth**, y, **mPaint**);  
 }  
 }

*//计算阅读百分比*  
 **float** fPercent = (**float**) (**m\_mbBufBegin** \* 1.0 / **m\_mbBufLen**);  
  */\*  
 以"#"补位  
 整数部分多了:不会截断,但是排在有效位最前面的0会被删除  
 整数部分少了:不作处理  
 小数部分多了:截断,建议指定RoundingMode,默认为RoundingMode.HALF\_EVEN  
 小数部分少了:不作处理  
 以"0"补位  
 整数部分多了:不会截断,排在有效位前面的0也不会被删除  
 整数部分少了:补0  
 小数部分多了:截断,建议指定RoundingMode,默认为RoundingMode.HALF\_EVEN  
 小数部分少了:补0  
 \*/* DecimalFormat df = **new** DecimalFormat(**"#0.0"**);  
 String strPercent = df.format(fPercent \* 100) + **"%"**;  
 **int** nPercentWidth = (**int**) **mPaint**.measureText(**"999.9%"**) + 1;  
 c.drawText(strPercent, **mWidth** - nPercentWidth, **mHeight** - 5, **mPaint**);

##### ※setBgBitmap

**public void** setBgBitmap(Bitmap BG)

参数：Bitmap背景图的位图。

作用：设置背景图片。

**m\_book\_bg** = BG;

##### ※isfirstPage

**public boolean** isfirstPage()

返回值：boolean

作用：判断是否是第一页（是返回true，不是返回false）。

**return m\_isfirstPage**;

##### ※islastPage

**public boolean** islastPage()

返回值：boolean

作用：判断是否是最后一页（是返回true，不是返回false）。

**return m\_islastPage**;

##### ※readParagraphBack

**protected byte**[] readParagraphBack(**int** nFromPos)

参数：**int**下一段落的开始的那个缓冲区指针。

返回值：装有上一段落内容的字节数组。

作用：读取上一段落的内容。

**int** nEnd = nFromPos;  
**int** i;  
**byte** b0, b1;  
**if** (**m\_strCharsetName**.equals(**"UTF-16LE"**)) {  
 i = nEnd - 2;  
 **while** (i > 0) {  
 b0 = **m\_mbBuf**.get(i);  
 b1 = **m\_mbBuf**.get(i + 1);  
 **if** (b0 == 0x0a && b1 == 0x00 && i != nEnd - 2) {  
 i += 2;  
 **break**;  
 }  
 i--;  
 }  
  
} **else if** (**m\_strCharsetName**.equals(**"UTF-16BE"**)) {  
 i = nEnd - 2;  
 **while** (i > 0) {  
 b0 = **m\_mbBuf**.get(i);  
 b1 = **m\_mbBuf**.get(i + 1);  
 **if** (b0 == 0x00 && b1 == 0x0a && i != nEnd - 2) {  
 i += 2;  
 **break**;  
 }  
 i--;  
 }  
} **else** {  
 i = nEnd - 1;  
 **while** (i > 0) {  
 b0 = **m\_mbBuf**.get(i);  
 **if** (b0 == 0x0a && i != nEnd - 1) { *//如果b0为换行并且i不为第一个字符* i++; *//i之前为换行，++之后就是下一段的第一个字符* **break**;  
 }  
 i--;  
 }  
}  
**if** (i < 0)  
 i = 0;  
**int** nParaSize = nEnd - i; *//得到段落大小***int** j;  
**byte**[] buf = **new byte**[nParaSize];  
**for** (j = 0; j < nParaSize; j++) {  
 buf[j] = **m\_mbBuf**.get(i + j);  
}  
**return** buf;

##### ※readParagraphForward

**protected byte**[] readParagraphForward(**int** nFromPos)

参数：**int**上一段落的末尾的那个缓冲区指针。

返回值：装有下一段落内容的字节数组。

作用：读取下一段落的内容。

**int** nStart = nFromPos;  
 **int** i = nStart;  
 **byte** b0, b1;  
 *// 根据编码格式判断换行* **if** (**m\_strCharsetName**.equals(**"UTF-16LE"**)) {  
 **while** (i < **m\_mbBufLen** - 1) {  
 b0 = **m\_mbBuf**.get(i++);  
 b1 = **m\_mbBuf**.get(i++);  
 **if** (b0 == 0x0a && b1 == 0x00) {  
 **break**;  
 }  
 }  
 } **else if** (**m\_strCharsetName**.equals(**"UTF-16BE"**)) {  
 **while** (i < **m\_mbBufLen** - 1) {  
 b0 = **m\_mbBuf**.get(i++);  
 b1 = **m\_mbBuf**.get(i++);  
 **if** (b0 == 0x00 && b1 == 0x0a) {  
 **break**;  
 }  
 }  
 } **else** {  
 **while** (i < **m\_mbBufLen**) {  
 b0 = **m\_mbBuf**.get(i++);  
 **if** (b0 == 0x0a) {  
 **break**;  
 }  
 }  
 }  
 **int** nParaSize = i - nStart;  
 **byte**[] buf = **new byte**[nParaSize];  
 **for** (i = 0; i < nParaSize; i++) {  
 buf[i] = **m\_mbBuf**.get(nFromPos + i);  
 }  
 **return** buf;

##### ※pageDown

**protected** Vector<String> pageDown()

返回值：Vector<String> 下一页每一行的内容所构成的String数组。

作用：读取下一页的内容。

String strParagraph = **""**;  
Vector<String> lines = **new** Vector<String>();  
**while** (lines.size() < **mLineCount** && **m\_mbBufEnd** < **m\_mbBufLen**) {  
 **byte**[] paraBuf = readParagraphForward(**m\_mbBufEnd**); *// 读取一个段落* **m\_mbBufEnd** += paraBuf.**length**;  
 **try** {  
 strParagraph = **new** String(paraBuf, **m\_strCharsetName**);  
 } **catch** (UnsupportedEncodingException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 }  
 String strReturn = **""**;  
 **if** (strParagraph.indexOf(**"\r\n"**) != -1) { *//回车换行* strReturn = **"\r\n"**;  
 strParagraph = strParagraph.replaceAll(**"\r\n"**, **""**); *//正则表达式 字符* } **else if** (strParagraph.indexOf(**"\n"**) != -1) {  
 strReturn = **"\n"**;  
 strParagraph = strParagraph.replaceAll(**"\n"**, **""**);  
 }  
 **if** (strParagraph.length() == 0) { *//段落长度为0添加到vector数组中* lines.add(strParagraph);  
 }  
 **while** (strParagraph.length() > 0) {  
 *//计算制定长度的字符串（字符长度、字符个数、显示的时候真实的长度）  
 //measureForwards为true,从头开始测否则从尾向前测  
 //maxWidth 最大的宽度  
 //measuredWidth实测长度* **int** nSize = **mPaint**.breakText(strParagraph, **true**, **mVisibleWidth**,  
 **null**);  
 lines.add(strParagraph.substring(0, nSize));  
 strParagraph = strParagraph.substring(nSize);  
 **if** (lines.size() >= **mLineCount**) {  
 **break**;  
 }  
 }  
 **if** (strParagraph.length() != 0) {  
 **try** {  
 **m\_mbBufEnd** -= (strParagraph + strReturn)  
 .getBytes(**m\_strCharsetName**).**length**;  
 } **catch** (UnsupportedEncodingException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 }  
 }  
}  
**return** lines;

##### ※pageUp

**protected void** pageUp()

作用：将缓冲区指针移动到上一页的上一页去。

**if** (**m\_mbBufBegin** < 0)  
 **m\_mbBufBegin** = 0;  
Vector<String> lines = **new** Vector<String>();  
String strParagraph = **""**;  
**while** (lines.size() < **mLineCount** && **m\_mbBufBegin** > 0) {  
 Vector<String> paraLines = **new** Vector<String>();  
 **byte**[] paraBuf = readParagraphBack(**m\_mbBufBegin**);  
 **m\_mbBufBegin** -= paraBuf.**length**;  
 **try** {  
 strParagraph = **new** String(paraBuf, **m\_strCharsetName**);  
 } **catch** (UnsupportedEncodingException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 }  
 strParagraph = strParagraph.replaceAll(**"\r\n"**, **""**);  
 strParagraph = strParagraph.replaceAll(**"\n"**, **""**);  
 **if** (strParagraph.length() == 0) {  
 paraLines.add(strParagraph);  
 }  
 **while** (strParagraph.length() > 0) {  
 **int** nSize = **mPaint**.breakText(strParagraph, **true**, **mVisibleWidth**,  
 **null**);  
 paraLines.add(strParagraph.substring(0, nSize));  
 strParagraph = strParagraph.substring(nSize);  
 }  
 lines.addAll(0, paraLines);  
}  
**while** (lines.size() > **mLineCount**) {  
 **try** {  
 **m\_mbBufBegin** += lines.get(0).getBytes(**m\_strCharsetName**).**length**;  
 lines.remove(0);  
 } **catch** (UnsupportedEncodingException e) {  
 *//* ***TODO Auto-generated catch block*** e.printStackTrace();  
 }  
}  
**m\_mbBufEnd** = **m\_mbBufBegin**;  
**return**;

##### ※prePage

**protected void** prePage()

**throws** IOException

作用：判断是否到达第一页（即书头）给m\_isfirstPage赋值。若没有到第一页就将上一页的内容放到m\_lines中。这里与nextPage略有区别的是这里调用pageUp()函数移动缓冲区指针，再去调用pageDown()读取内容。

**if** (**m\_mbBufBegin** <= 0) {  
 **m\_mbBufBegin** = 0;  
 **m\_isfirstPage**=**true**;  
 **return**;  
 }**else  
 m\_isfirstPage**=**false**;  
 **m\_lines**.clear();  
 pageUp();  
 **m\_lines** = pageDown();  
}

## §4 其他类详解

### §4.1 Path类

public class Path  
extends Object

|  |  |
| --- | --- |
| java.lang.Object | |
| ↳ | android.graphics.Path |

Path类封装由直线段，二次曲线和三次曲线组成的复合（多个轮廓）几何路径。可以使用canvas.drawPath（path，paint）进行填充或描边绘制（基于Paint的Style），也可以用于剪切或在路径上绘制文本。

### §4.2 Bitmap类

public final class Bitmap  
extends Object implements Parcelable

|  |  |
| --- | --- |
| java.lang.Object | |
| ↳ | android.graphics.Bitmap |

可以把他看作一个画架，可以把画放到上面，然后可以进行一些处理，比如获取图像文件信息，做旋转切割，放大缩小等操作。

### §4.3 PointF类

public class PointF  
extends Object implements Parcelable

|  |  |
| --- | --- |
| java.lang.Object | |
| ↳ | android.graphics.PointF |

PointF拥有两个浮点坐标（即一个点的坐标，且横坐标和纵坐标都为浮点数）

属性：

public float x;

public float y;

### §4.4 Paint类

public class Paint  
extends Object

|  |  |
| --- | --- |
| java.lang.Object | |
| ↳ | android.graphics.Paint |

画笔类主要保存了颜色、样式，绘制几何图形、文本和位图的样式和颜色等绘制信息，指定了如何绘制文本和图形，画笔对象有很多设置方法，大体上可以分为两类，一类与图形绘制相关，一类与文本绘制相关。

Android在用画笔的时候有三种Style，分别是  
Paint.Style.STROKE 只绘制图形轮廓（描边）  
Paint.Style.FILL 只绘制图形内容  
Paint.Style.FILL\_AND\_STROKE 既绘制轮廓也绘制内容

### §4.5 Scroller类

public class Scroller  
extends Object

|  |  |
| --- | --- |
| java.lang.Object | |
| ↳ | android.widget.Scroller |

此类封装了滚动。您可以使用滚动条（Scroller 或OverScroller）收集产生滚动动画所需的数据，例如，响应于挥动手势。滚动条会随着时间的推移为您跟踪滚动偏移量，但不会自动将这些位置应用于您的视图。您获得新坐标的速率并加以利用可使滚动动画看起来平滑。

该类监听用户的触摸事件,并监听传入的坐标,来实现滑动

Scroller只是个计算器，提供插值计算，让滚动过程具有动画属性，但它并不是UI，也不是辅助UI滑动，反而是单纯地为滑动提供计算。该类最主要的作用是为偏移过程制定一定的控制流程，从而使偏移更流畅，更完美。

### §4.6 GradientDrawable类

public class GradientDrawable  
extends Drawable

|  |  |  |
| --- | --- | --- |
| java.lang.Object | | |
| ↳ | android.graphics.drawable.Drawable | |
|  | ↳ | android.graphics.drawable.GradientDrawable |

具有颜色渐变的Drawable，用于按钮，背景等。

GradientDrawable 表示一个渐变区域，可以实现线性渐变、发散渐变和平铺渐变效果。

可以使用<shape>元素在XML文件中定义它。

### §4.7 MappedByteBuffer类

public abstract class MappedByteBuffer  
extends ByteBuffer

|  |  |  |  |
| --- | --- | --- | --- |
| java.lang.Object | | | |
| ↳ | java.nio.Buffer | | |
|  | ↳ | java.nio.ByteBuffer | |
|  |  | ↳ | java.nio.MappedByteBuffer |

MappedByteBuffer 是Java NIO中引入的一种硬盘物理文件和内存映射方式，当物理文件较大时，采用MappedByteBuffer，读写性能较高，其内部的核心实现是DirectByteBuffer(JVM 堆外直接物理内存)。

一个直接字节缓冲区，其内容是文件的内存映射区域。

映射的字节缓冲区是通过FileChannel.map方法创建的。ByteBuffer该类通过特定于内存映射文件区域的操作扩展了该类。

映射的字节缓冲区及其表示的文件映射将保持有效，直到缓冲区本身被垃圾回收为止。

映射的字节缓冲区的内容可以随时更改，例如，如果此程序或其他程序更改了映射文件的相应区域的内容。此类更改是否发生以及何时发生，取决于操作系统，因此未指定。

映射字节缓冲区的全部或部分可能随时无法访问，例如，如果映射文件被截断。尝试访问映射的字节缓冲区的不可访问区域将不会更改缓冲区的内容，并且将导致在访问时或稍后发生未指定的异常。因此，强烈建议采取适当的预防措施，以防止该程序或并发运行的程序对映射文件的操作，除非要读取或写入文件的内容。否则，映射的字节缓冲区的行为与普通的直接字节缓冲区没有什么不同。

### §4.8 ColorMatrix类

public class ColorMatrix  
extends [Object](https://developer.android.google.cn/reference/java/lang/Object)

|  |  |
| --- | --- |
| java.lang.Object | |
| ↳ | android.graphics.ColorMatrix |

相当于颜色滤镜。

颜色矩阵，对每一个像素点的颜色（ARGB）进行更改，像ps一样的改变图像的颜色。因为运算大量的像素点，colorMatrix是native方法。

是一个4x5矩阵，用于转换位图的颜色和alpha分量。

### §4.9 ColorMatrixColorFilter类

public class ColorMatrixColorFilter  
extends [ColorFilter](https://developer.android.google.cn/reference/android/graphics/ColorFilter)

|  |  |  |
| --- | --- | --- |
| java.lang.Object | | |
| ↳ | android.graphics.ColorFilter | |
|  | ↳ | android.graphics.ColorMatrixColorFilter |

颜色过滤器，提供颜色变化之类的效果。

通过4x5颜色矩阵转换颜色的滤色器。该滤镜可用于更改像素的饱和度，从YUV转换为RGB等。

### §4.10 RandomAccessFile类

public class RandomAccessFile  
extends Object implements DataOutput, DataInput, Closeable

|  |  |
| --- | --- |
| java.lang.Object | |
| ↳ | java.io.RandomAccessFile |

此类的实例支持读取和写入随机访问文件。随机访问文件的行为类似于存储在文件系统中的大字节数组。隐式数组中有一种游标或索引，称为文件指针；输入操作从文件指针开始读取字节，然后使文件指针越过读取的字节。如果随机访问文件是在读/写模式下创建的，则输出操作也可用；输出操作从文件指针开始写入字节，然后将文件指针向前移动超过写入的字节。写入超出隐含数组当前端的输出操作将导致数组扩展。该getFilePointer方法可以读取文件指针，并可以通过该方法设置文件指针seek。

通常，对于此类中的所有读取例程，如果在读取所需的字节数之前已到达文件末尾，则将抛出EOFException（是一种 IOException）。如果由于文件末尾以外的任何其他原因而无法读取任何字节，则抛出一个IOException非字节EOFException。特别是，IOException如果流已关闭，则可能引发。

### §4.11 DecimalFormat类

public class DecimalFormat  
extends NumberFormat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| java.lang.Object | | | | |
| ↳ | java.text.Format | | | |
|  | ↳ | android.icu.text.UFormat | | |
|  |  | ↳ | android.icu.text.NumberFormat | |
|  |  |  | ↳ | android.icu.text.DecimalFormat |

格式化类。具有多种功能，旨在可以在任何语言环境中解析和格式化数字，包括对西方，阿拉伯或印度数字的支持。它支持不同类型的数字，包括整数（“123”），浮点数字（“123.4”），科学计数法（“1.23E4”），百分比（“12％”）和货币金额（“$ 123.00”， “USD123.00”，“123.00 US dollars”）。所有这些风格都可以轻松定制。

例如：**"#0.0"**

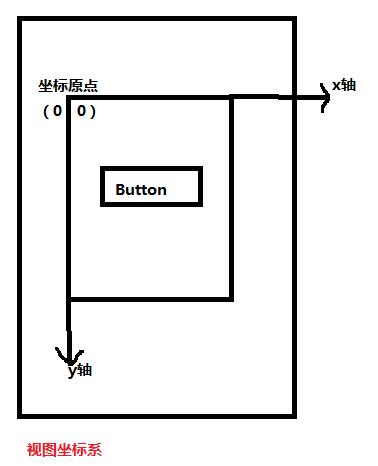
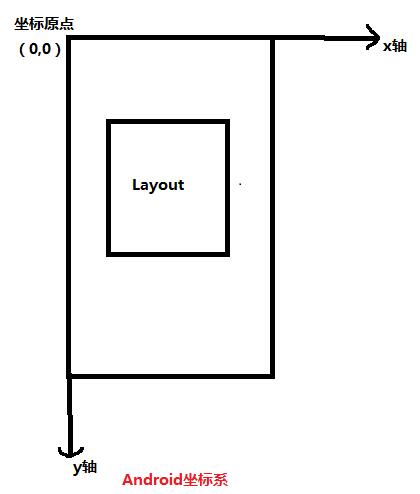
*以"#"补位  
 整数部分多了:不会截断,但是排在有效位最前面的0会被删除  
 整数部分少了:不作处理  
 小数部分多了:截断,建议指定RoundingMode,默认为RoundingMode.HALF\_EVEN  
 小数部分少了:不作处理  
 以"0"补位  
 整数部分多了:不会截断,排在有效位前面的0也不会被删除  
 整数部分少了:补0  
 小数部分多了:截断,建议指定RoundingMode,默认为RoundingMode.HALF\_EVEN  
 小数部分少了:补0*

## §5 其他要点

### §5.1 安卓坐标系与视图坐标系

※安卓坐标系：在安卓中,屏幕左上角为Android坐标系的原点（如下左图）,这个点以右边边和下边为正方向,在触碰事件中,getRawX和getRawY是获得Android坐标系的坐标。

※视图坐标系：视图坐标系描述了子视图在父视图中的位置,和安卓坐标系相同,他也是以右边和下边为正方向（如下右图）,在触碰事件中,getX和getY获得的是视图坐标系的XY坐标.我们也可以调用View的方法,来获取坐标。



### §5.2 View类的要点

View是Android中所有控件的基类，不管是简单的TextView，Button还是复杂的LinearLayout和ListView，它们的共同基类都是View；  
View是一种界面层的控件的一种抽象，它代表了一个控件，除了View还有ViewGroup，从名字来看ViewGroup可以翻译为控件组，即一组View；  
在Android中，ViewGroup也继承了View，这就意味着View可以是单个控件，也可以是由多个控件组成的一组控件。

#### §5.2.1 invalidate()和postInvalidate()区分

invalidate()在主线程、UI线程中调用。  
postInvalidate()在工作者线程，在非UI线程中调用。

把主要是负责控制UI界面的显示、更新和控件交互的线程称为UI线程。

#### §5.2.2 View的三大流程

measure：测量，决定了View的宽度和高度

layout：布局，决定了View最终的宽高及四个顶点的位置

draw：绘制，将View绘制到屏幕上

#### §5.2.3 onDraw()的调用时间

初始化后被调用一次；

invaildate()的时候会被调用；

postinvaildate()的时候会被调用。

### §5.3 Canvas类的要点

方法介绍

* public boolean clipPath (Path path)
* public boolean clipPath (Path path, Region.Op op)
* public boolean clipRect (Rect rect)
* public boolean clipRect (Rect rect, Region.Op op)

这些方法用于在画布上裁剪对应的区域（Rect是一个矩形，Path是一个自定义路径）。另外一个参数是Region.Op ,它规定了用不同的方式来剪辑区域。设**上一次裁剪区域**为A，**当前裁剪区域**为B：

* DIFFERENCE ：裁剪差异部分，具体为A - A∩B
* INTERSECT：裁剪两者相交的部分，A∩B
* UNION：裁剪两者所有区域，A∪B
* XOR：exclusive OR的简写，翻译为异或，也就是两者重叠部分为0，不重叠部分为1，自然裁剪出来的就是不重叠的部分，也就是A∪B - A∩B
* REVERSE\_DIFFERENCE：裁剪差异部分，具体为B - A∩B
* REPLACE：B的裁剪区域会覆盖A的区域，B  
  当方法不指定裁剪模式时，默认为INTERSECT

**注：**上一次裁剪区域是在调用最新的clipxxx之前所形成的区域。

### §5.4 computeScroll和Scroller滑动事件的处理

Scroller只是个计算器，提供插值计算，让滚动过程具有动画属性。它并不是UI，也不是滑动辅助UI运动，只是单纯地为滑动提供计算。

computeScroll也不是来让ViewGroup滑动的，真正让ViewGroup滑动的是scrollTo,scrollBy。computeScroll的作用是计算ViewGroup如何滑动。而computeScroll是通过draw来调用。

computeScroll和Scroller都是计算，但两者没有关系。computeScroll和Scroller要是非得拉关系的话，那就是computeScroll可以参考Scroller计算结果来影响scrollTo,scrollBy；从而使得滑动发生改变。也就是Scroller不会调用computeScroll，反而是computeScroll调用Scroller。

computeScroll调用Scroller，只要computeScroll调用连续，Scroller也会连续，实质上computeScroll的连续性又由invalidate方法控制；scrollTo,scrollBy都会调用invalidate，而invalidate回去触发draw,从而computeScroll被连续调用。所以，Scroller会被连续调用，除非invalidate停止调用。

为了不重复影响scrollTo,scrollBy，那么Scroller必须终止计算currX，currY。要知道计算有没有终止，需要通过mScroller.computeScrollOffset()。

### §5.5 JAVA设计模式之工厂模式—Factory Pattern

工厂模式用于对象的创建，使得客户从具体的产品对象中被解耦。

通过工厂模式可以实现创建者和调用者的分离，实现代码之间解耦，依赖。一句大白话就是就是减少类与类之间的依赖关系。

分类：**简单工厂模式、工厂模式、抽象工厂模式**

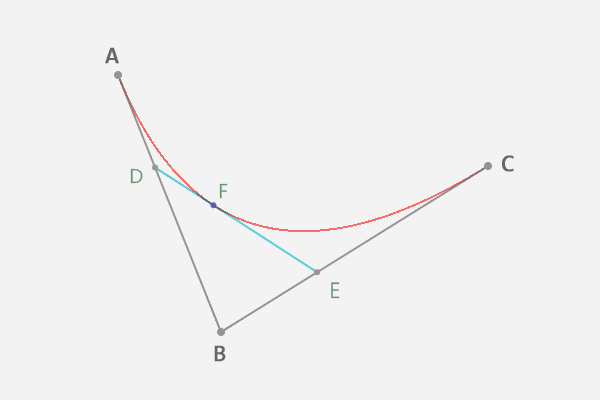
### §5.6 贝塞尔曲线的理解

Bezier曲线是用一系列点控制曲线状态的。主要分为

数据点：确定曲线的起始和结束位置

控制点：确定曲线的弯曲程度

**一个控制点的情况，如图所示：**

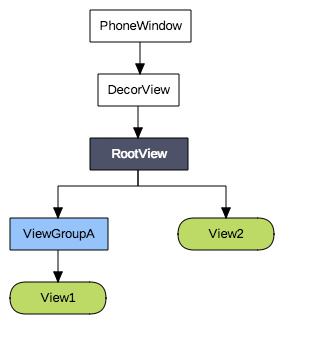


1. A/B/C三点是确定的
2. 在AB上任取一点D，得到ratio = AD/AB
3. 再由BE/BC = ratio 得到E点
4. 连接DE，同理DF/DE = ratio得到F
5. 而F点就是曲线上的一点，当然凭着这一点是无法得到整条曲线的
6. 于是，再来一遍，重新取D点得到新的F点，以此类推。

多个控制点的情况类似。

### §5.7 事件分发机制原理

**安卓上面的View是树形结构的（如图）。**

****

基于树形结构，事件可以进行有序的分发。

事件收集之后最先传递给 Activity，然后依次向下传递，流程如下：

Activity －> PhoneWindow －> DecorView －> ViewGroup －> ... －> View

如果最后分发到View，如果这个View也没有处理事件，那么这个事件会按照反方向回传，最终传回给Activity，如果最后 Activity 也没有处理，本次事件才会被抛弃:

Activity <－ PhoneWindow <－ DecorView <－ ViewGroup <－ ... <－ View

**责任链模式。**

### §5.8 绘图的几个要素

* **Drawable：**通用的图形对象，用于装载常用格式的图像，既可以是PNG，JPG这样的图像， 也是前面学的那13种Drawable类型的可视化对象！我们可以理解成一个用来放画的——画框！
* **Bitmap(位图)：**我们可以把他看作一个画架，我们先把画放到上面，然后我们可以 进行一些处理，比如获取图像文件信息，做旋转切割，放大缩小等操作！
* **Canvas(画布)：**如其名，画布，我们可以在上面作画(绘制)，你既可以用Paint(画笔)， 来画各种形状或者写字，又可以用Path(路径)来绘制多个点，然后连接成各种图形！
* **Matrix(矩阵)：**用于图形特效处理的，颜色矩阵(ColorMatrix)，还有使用Matrix进行图像的 平移，缩放，旋转，倾斜等！