Android程序设计

绘制原理

2019.6.11

isszym sysu.edu.cn

官方文档（中文） 官方文档（英文） runoob cnblogs

主目录

- View的绘制概述
- performTraversals
- measure、layout和draw
- invalidate
- requestLayout
- 自定义视图

View的绘制和渲染原理

参考

View的绘制概述

performTraversals

measure、layout和draw

Invalidate

requestLayout

自定义视图

View的绘制概述

参考1 参考2 参考3

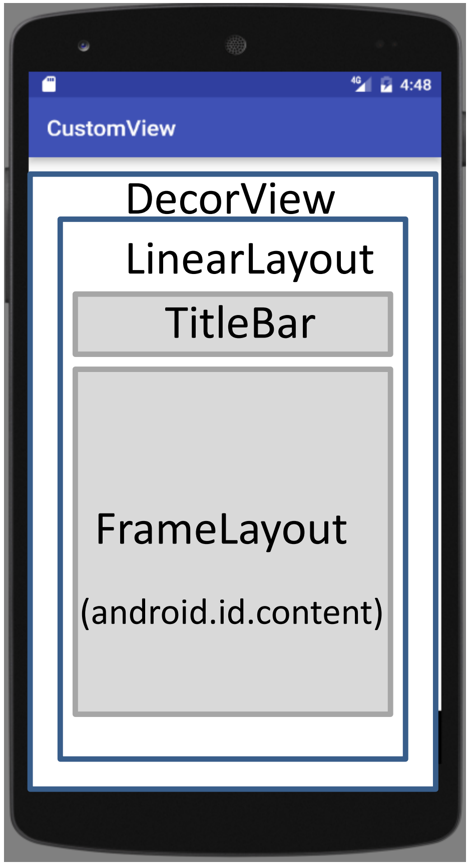
- DecorView是每个Activity界面的根视图。

（1）DecorView是FrameLayout的子类，也是PhoneWindow对象的内部类。DecorView对象的创建是在setContentView方法第一次执行的时候创建的。

（2）DecorView里面包含一个线性布局，系统会根据主题选择该线性布局。例如，当设置是no-title的主题，PhoneWindow就会选择一个没有title的线性布局生成DecorView的子View。

（3）用findViewById获得id为content的FrameLayout对象，即contentView。**setContentView**就是将编写的View或者Xml布局加入到这个id为content的FrameLayout里面。

（4）这里也是为什么requestWindowFeature方法（获得应用标题等）要在setContentView 方法之前调用的原因。

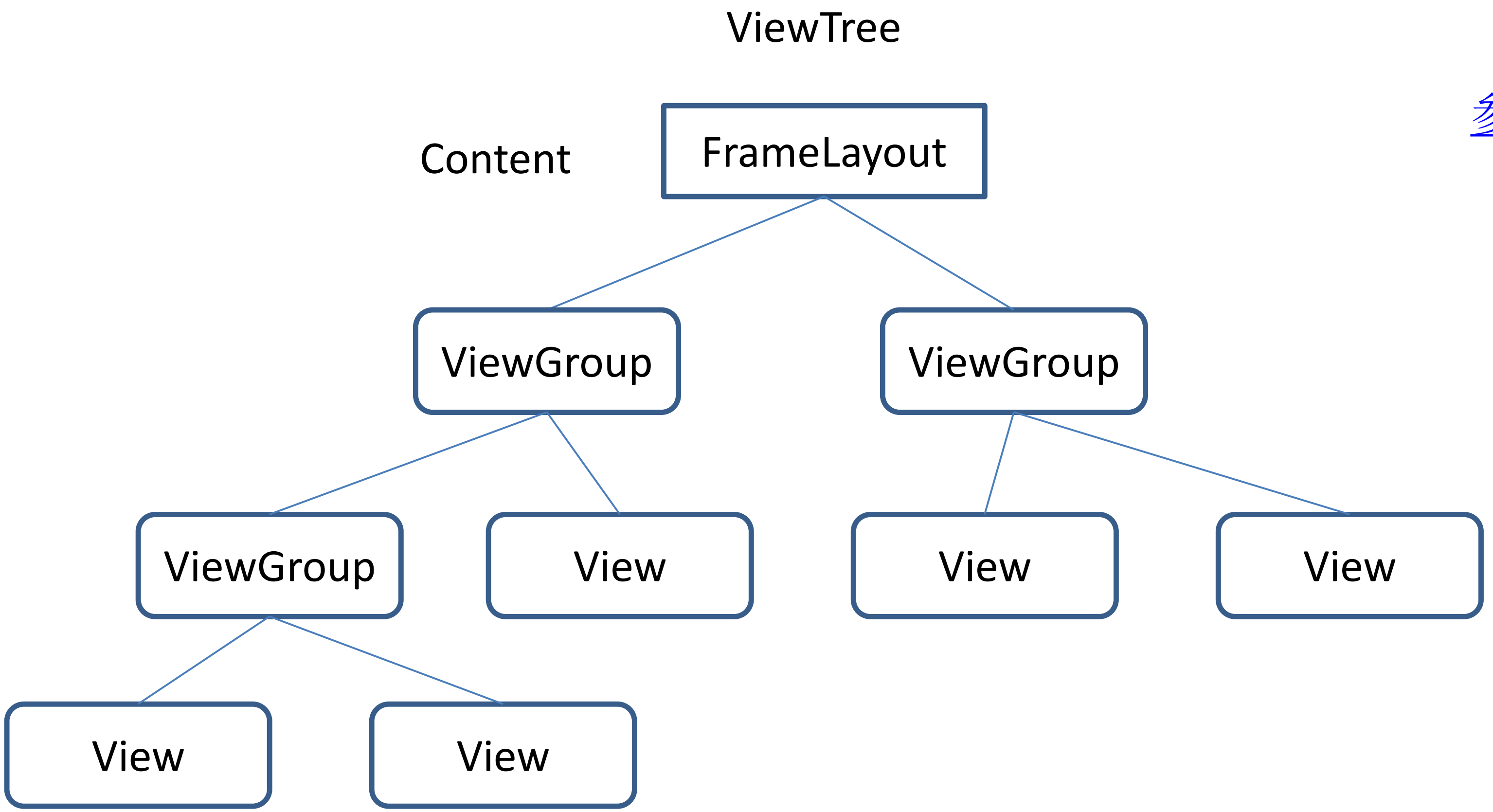


这是创建DecorView对象可选择的一个系统主题：

R.layout.screen_simple

```xml
<?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:fitsSystemWindows="true"
    android:orientation="vertical">
    <ViewStub android:id="@+id/action_mode_bar_stub"
        android:inflatedId="@+id/action_mode_bar"
        android:layout="@layout/action_mode_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="?attr/actionBarTheme" />
    <FrameLayout
        android:id="@android:id/content"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:foregroundInsidePadding="false"
        android:foregroundGravity="fill_horizontal|top"
        android:foreground="?android:attr/windowContentOverlay" />
</LinearLayout>
```

- 执行setContentView将遍历布局文件中的控件树进行绘制



参考

```
public void setContentView(@LayoutRes int layoutResID) {
    getWindow().setContentView(layoutResID);
    initWindowDecorActionBar();
}
```

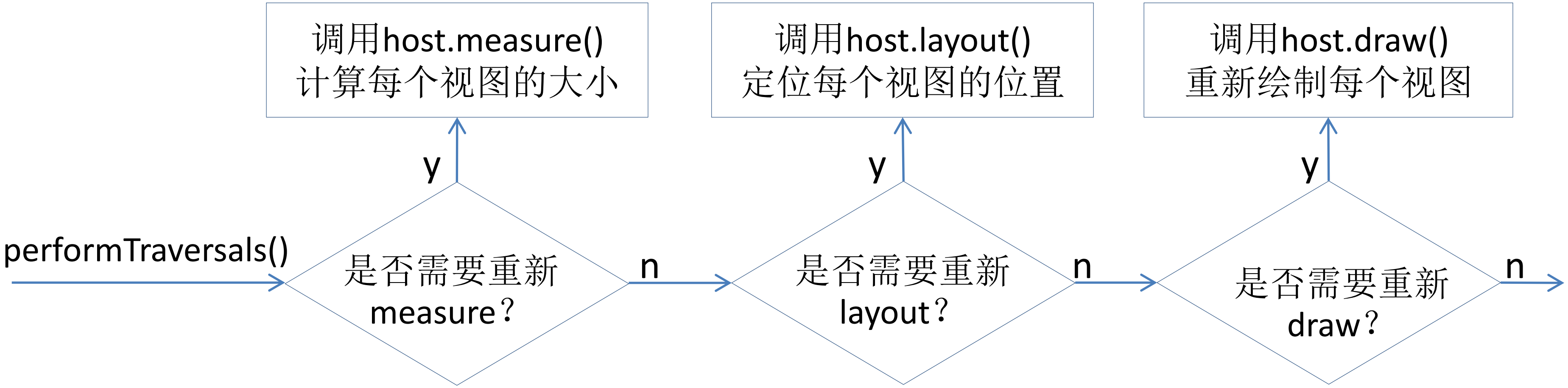
getWindow()得到PhoneWindow对象

performTraversals

参考　参考

执行setContentView()后经过多层调用（见后续源码），最后会调用到方法performTraversals()。该方法会从根DecorView开始遍历当前window中所有的view，并根据条件，例如，mLayoutRequested=true，确定是否要调用它们的measure方法、layout方法和draw方法，以测量view的大小、为它们在父view进行定位、以及重新绘制它们。

这些方法都是view的内部方法，在这些方法中会调用onMeasure方法、onLayout()和onDraw()方法三个可覆盖方法，使用户可以通过覆盖这些方法获取View的大小测量和定位，以及在View上进行绘制。具体的描述见下节。



PhoneWindow的setContentView部分源码:

```
@Override
public void setContentView(int layoutResID) {
    if (mContentParent == null) {
        installDecor();
    } else if (!hasFeature(FEATURE_CONTENT_TRANSITIONS)) {
        mContentParent.removeAllViews();
    }

    if (hasFeature(FEATURE_CONTENT_TRANSITIONS)) {
        final Scene newScene = Scene.getSceneForLayout(mContentParent, layoutResID,
                getContext());
        transitionTo(newScene);
    } else {
        mLayoutInflater.inflate(layoutResID, mContentParent);
    }
    mContentParent.requestApplyInsets();
    final Callback cb = getCallback();
    if (cb != null && !isDestroyed()) {
        cb.onContentChanged();
    }
    mContentParentExplicitlySet = true;
}
```

mContentParent为前面的FrameLayout

Inflater简单来说就是将要inflate的View的属性（包括layout_width和layout_height）解析出来设置到View里，并将这个View添加到root这个ViewGroup的列表中。

用于请求绘制，再经过几层调用之后会调用ViewRootImpl的requestFitSystemWindows()。

ViewRootImpl的requestFitSystemWindows：

```
@Override
public void requestFitSystemWindows() {
    checkThread();
    mApplyInsetsRequested = true;
    scheduleTraversals();
}

void scheduleTraversals() {
    if (!mTraversalScheduled) {
        mTraversalScheduled = true;
        //暂停了handler的后续消息处理，防止界面刷新的时候出现同步问题
        mTraversalBarrier = mHandler.getLooper().getQueue().postSyncBarrier();
        //将runnable发送给handler执行
        mChoreographer.postCallback(Choreographer.CALLBACK_TRAVERSAL,
                                                     mTraversalRunnable, null);
        if (!mUnbufferedInputDispatch) {
            scheduleConsumeBatchedInput();
        }
        notifyRendererOfFramePending();
        pokeDrawLockIfNeeded();
    }
}
```

```
final TraversalRunnable mTraversalRunnable = new TraversalRunnable();

final class TraversalRunnable implements Runnable {
    @Override public void run() {
        doTraversal();
    }
}
void doTraversal() {
    if (mTraversalScheduled) {
        mTraversalScheduled = false;
        mHandler.getLooper().getQueue().removeSyncBarrier(mTraversalBarrier);
        if (mProfile) {
            Debug.startMethodTracing("ViewAncestor");
        }
        performTraversals();
        if (mProfile) {
            Debug.stopMethodTracing();
            mProfile = false;
        }
    }
}
```

ViewRootImpl的performTraversals

performTraversals()中调用performMeasure()确定了ViewTree中的每一个View的width和height，然后调用performLayout()确定这些View在屏幕的位置，最后调用performDraw()把每一个View画在屏幕上的相应位置上就可以了。

```
private void performTraversals() {
    final View host = mView;//mView就是DecorView根布局,是ViewRootImpl管理的View树
                            //    的根节点，final修饰，避免运行过程中修改
    ......
    //最外层的根视图的widthMeasureSpec和heightMeasureSpec由来
    //lp.width和lp.height在创建ViewGroup实例时等于MATCH_PARENT
    int childWidthMeasureSpec = getRootMeasureSpec(mWidth, lp.width);
    int childHeightMeasureSpec = getRootMeasureSpec(mHeight, lp.height);
    ......
    performMeasure(childWidthMeasureSpec, childHeightMeasureSpec);
    ......
    performLayout(lp,windowWidth,windowHeight);
    ......
    performDraw();
    ......
}
```

measure、layout和draw

ViewRootImpl的performMeasure

performMeasure调用DecorView的measure方法，之后DecorView会遍历调用自己的ChildView的measure，ChildView如果是ViewGroup将遍历自己的ChidView，以此遍历整个ViewTree。

measure方法中调用onMeasure()。不同的ViewGroup的onMeasure()方法不同。FramLayout的onMeasure()遍历了自己的Child，找到最宽的宽度和最高的高度，传到下一级测量。而View只要测量自己的大小就可以了。

当DecorView根节点下的View全部测量完毕，各个View的变量mMeasuredWidth和mMeasuredHeight都是已知的了。

```
private void performMeasure(int childWidthMeasureSpec, int childHeightMeasureSpec){
    Trace.traceBegin(Trace.TRACE_TAG_VIEW, "measure");
    try {
        mView.measure(childWidthMeasureSpec, childHeightMeasureSpec);
    } finally {
        Trace.traceEnd(Trace.TRACE_TAG_VIEW);
    }
}
```

参考

ViewRootImpl的performLayout

performLayout也是由DecorView自上而下调用layout()，layout()调用onLayout()，直到调用到叶子节点，就可以将所有的View位置确定好了。

```
public void performLayout(LayoutParams lp, int windowWidth, int windowWidth) {
    …
    host.layout(0, 0, host.getMeasuredWidth(), host.getMeasuredHeight());
    …
}

public void layout(int l, int t, int r, int b) {
    if ((mPrivateFlags3 & PFLAG3_MEASURE_NEEDED_BEFORE_LAYOUT) != 0) {
        onMeasure(mOldWidthMeasureSpec, mOldHeightMeasureSpec);
        mPrivateFlags3 &= ~PFLAG3_MEASURE_NEEDED_BEFORE_LAYOUT;
    }
    int oldL = mLeft; int oldT = mTop; int oldB = mBottom; int oldR = mRight;
    boolean changed = isLayoutModeOptical(mParent) ?
            setOpticalFrame(l, t, r, b) : setFrame(l, t, r, b);
    ……
    if (changed || (mPrivateFlags & PFLAG_LAYOUT_REQUIRED)
                                        == PFLAG_LAYOUT_REQUIRED) {
        onLayout(changed, l, t, r, b);
    }
    ……
}
```

```
@Override
protected void onLayout(boolean changed, int left, int top, int right, int bottom){
    layoutChildren(left, top, right, bottom, false /* no force left gravity */);
}

void layoutChildren(int left, int top, int right,
                    int bottom, boolean forceLeftGravity){
    final int count = getChildCount();
    final int parentLeft = getPaddingLeftWithForeground();
    final int parentRight = right - left - getPaddingRightWithForeground();

    final int parentTop = getPaddingTopWithForeground();
    final int parentBottom = bottom - top - getPaddingBottomWithForeground();

    for (int i = 0; i < count; i++) {
        final View child = getChildAt(i);
        ……
        switch (absoluteGravity & Gravity.HORIZONTAL_GRAVITY_MASK) {
               case Gravity.CENTER_HORIZONTAL:
              …
        }
        child.layout(childLeft, childTop, childLeft + width, childTop + height);
   }
}
```

ViewRootImpl的performDraw

performDraw()由DecorView自上而下调用draw()，draw()调用onDraw()在屏幕上根据前面得到的尺寸和位置绘制View，并调用所有子节点的draw()，这样一直下去直到叶子结点，即普通View。

```
public void performDraw(){
    ……
    mView.draw(canvas);
    ……
}
public void  draw(Canvas  canvas){
    onDraw(canvas);
    for (int i = 0; i < childrenCount; i++) {
        ……

       drawChild(canvas, transientChild, drawingTime);
    }

}
protected boolean drawChild(Canvas canvas, View child, long drawingTime) {
    return child.draw(canvas, this, drawingTime);
}
```

invalidate

对一个view执行invalidate()，会引发ViewRootImpl执行scheduleTraversals()，从而执行该view的onDraw方法。View是可见时invalidate才有效。在UI主线程中使用invalidate方法，在其他线程中用postInvalidate方法。

```
public void invalidate() {
    invalidate(true);
}
void invalidate(boolean invalidateCache) {
  invalidateInternal(0, 0, mRight - mLeft, mBottom - mTop, invalidateCache, true);
}
void invalidateInternal(int l, int t, int r, int b, boolean invalidateCache,
                        boolean fullInvalidate) {
    ......
    // Propagate the damage rectangle to the parent view.
    final AttachInfo ai = mAttachInfo;
    final ViewParent p = mParent;
    if (p != null && ai != null && l < r && t < b) {
        final Rect damage = ai.mTmpInvalRect;
        damage.set(l, t, r, b);
        p.invalidateChild(this, damage); //设置刷新区域
    }
    ......
}
```

```
public final void invalidateChild(View child, final Rect dirty) {
    ViewParent parent = this;
    final AttachInfo attachInfo = mAttachInfo;
    ......
    do {
        ......
        //循环层层上级调运，直到ViewRootImpl会返回null
        parent = parent.invalidateChildInParent(location, dirty);
        ......
    } while (parent != null);
}
```

```
public  void invalidateChildInParent(int[]location, Rectdirty) {
        ......
       //View调运invalidate最终层层上传到ViewRootImpl后最终触发了该方法
        scheduleTraversals();
        ......
        return null;
}
```

由于此时视图没有设置重新测量的标志位，而且大小也没有变化，所以invalidate()只会使draw的遍历得到执行，而measure和layout都不会被执行。要把视图的绘制流程完整走一遍，要调用requestLayout() 。

postInvalidate最后还是在UI主线程中调用了View的invalidate方法，最后实现View的绘制流程。

```
public void postInvalidate() {
    postInvalidateDelayed(0);
}
public void postInvalidateDelayed(long delayMilliseconds) {
    final AttachInfo attachInfo = mAttachInfo;
    if (attachInfo != null) {
      attachInfo.mViewRootImpl.dispatchInvalidateDelayed(this, delayMilliseconds);
    }
}
public void dispatchInvalidateDelayed(View view, long delayMilliseconds) {
    Message msg = mHandler.obtainMessage(MSG_INVALIDATE, view);
    mHandler.sendMessageDelayed(msg, delayMilliseconds);
}
public void handleMessage(Message msg) {
    ......
    switch (msg.what) {
        case MSG_INVALIDATE:
            ((View) msg.obj).invalidate();
            break;
        ......
    }
    ......
}
```

requestLayout

- 调用View类的requestLayout将会遍历所有view的measure、layout和draw方法，并且不会像invalidate一样跳过一些方法。
- 下面为ViewRootImpl中的requestLayout方法。一个View的requestLayout将调用其上层View的requestLayout()，最终会调用ViewRootImpl中的requestLayout方法。
- 由于mLayoutRequested被设置成true， scheduleTraversals方法会遍历所有的measure、layout和draw方法。

```
@Override
public void requestLayout() {
    if (!mHandlingLayoutInLayoutRequest) {
        checkThread();
        mLayoutRequested = true;
        scheduleTraversals();
    }
}
```

View的requestLayout方法：

```
@CallSuper
public void requestLayout() {
    if (mMeasureCache != null) mMeasureCache.clear();
    if (mAttachInfo != null && mAttachInfo.mViewRequestingLayout == null) {
        //Only trigger request-during-layout logic if this is the view requesting it,
        // not the views in its parent hierarchy
        ViewRootImpl viewRoot = getViewRootImpl();
        if (viewRoot != null && viewRoot.isInLayout()) {
            if (!viewRoot.requestLayoutDuringLayout(this)) {
                return;
            }
        }
        mAttachInfo.mViewRequestingLayout = this;
    }
    mPrivateFlags |= PFLAG_FORCE_LAYOUT;
    mPrivateFlags |= PFLAG_INVALIDATED;
    if (mParent != null && !mParent.isLayoutRequested()) {
        mParent.requestLayout();
    }
    if (mAttachInfo != null && mAttachInfo.mViewRequestingLayout == this) {
        mAttachInfo.mViewRequestingLayout = null;
    }
}
```

自定义视图

参考

- 自定义视图可以继承View、TextView、ImageView、Button等控件和ViewGroup、LinearLayout、FrameLayout、RelativeLayout 等容器，可以为他们增加属性，绘制屏幕、响应消息，可以自定义回调函数。
- View的绘制由measure()、layout()、draw()完成。measure()用于计算视图大小，layouy()用于视图在屏幕中的位置（布局）， draw()用于绘制视图。
- View关于measure的方法：

```
public final void measure(int widthMeasureSpec, int heightMeasureSpec)
protected final void setMeasuredDimension(int measuredWidth,
                                          int measuredHeight)
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
```

- View关于layout的方法：

```
public void layout(int l, int t, int r, int b)
protected boolean setFrame(int left, int top, int right, int bottom)
protected void onLayout(boolean changed, int left, int top,
                                        int right, int bottom)
```

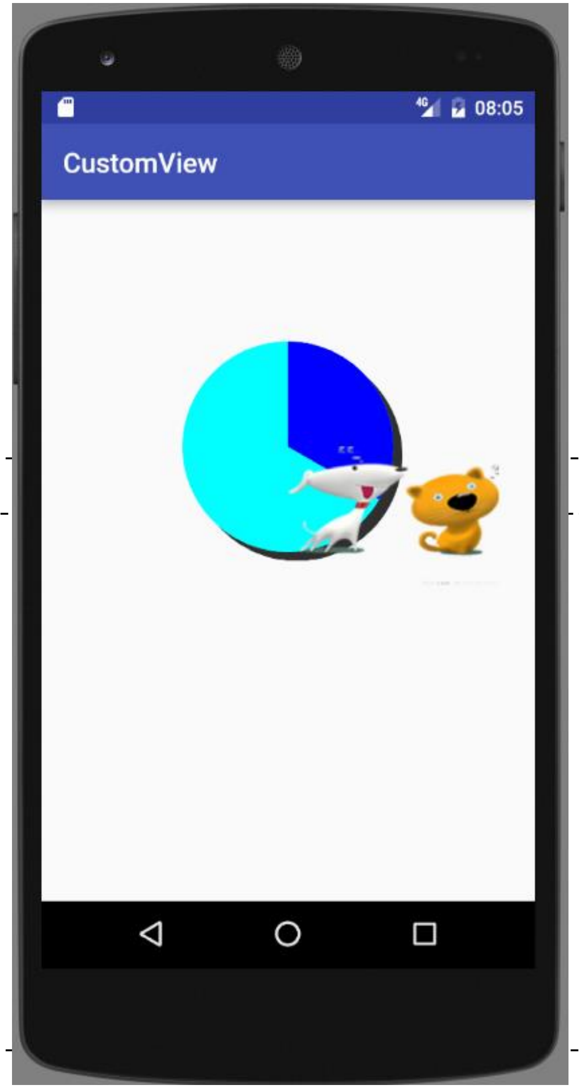
- View关于canvas的方法：

```
public void draw(Canvas canvas)
protected void onDraw(Canvas canvas)
```

*关于View的这些方法介绍见课件（二）。

acitvity_main.xml

```
<com.example.isszym.customview.CustomView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    app:shadowColor="#FF303030"
    app:shadowDx="20"
    app:shadowDy="20"
    app:shadowRadius="20.0"
    app:src="@drawable/dogs" />
```



attr.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="Shadow">
        <attr name="src" format="reference" />
        <attr name="shadowDx" format="integer" />
        <attr name="shadowDy" format="integer" />
        <attr name="shadowColor" format="color"/>
        <attr name="shadowRadius" format="float"/>
    </declare-styleable>
</resources>
```

CustomView.xml

```java
public class CustomView extends View {
    private final static String TAG = CustomView.class.getSimpleName();
    private Paint mPaint;
    private RectF oval;
    int mDx, mDy;
    float mRadius;
    int mShadowColor;
    int mBitmapID;
    Context mContext;

    public CustomView(Context context) throws Exception  {
        super(context);  mContext=context;
    }

    public CustomView(Context context, AttributeSet attrs) throws Exception  {
        super(context, attrs); init(context,attrs);
    }

    public CustomView(Context context, AttributeSet attrs, int defStyleAttr)
          throws Exception {
        super(context, attrs, defStyleAttr);   init(context,attrs);
    }
```

```
private void init(Context context,AttributeSet attrs) throws Exception {
        mContext=context;
        mPaint = new Paint();
        mPaint.setAntiAlias(true);
        oval=new RectF();
        /**  提取属性定义   */
        TypedArray typedArray = context.obtainStyledAttributes(attrs,
                                                               R.styleable.Shadow);
        mBitmapID = typedArray.getResourceId(R.styleable.Shadow_src,-1);
        if (mBitmapID == -1){throw new Exception("Shadow的Src属性必须是图像ID"); }
        mDx = typedArray.getInt(R.styleable.Shadow_shadowDx,21);
        mDy = typedArray.getInt(R.styleable.Shadow_shadowDy,21);
        mRadius = typedArray.getFloat(R.styleable.Shadow_shadowRadius,21);
        mShadowColor = typedArray.getColor(R.styleable.Shadow_shadowColor,Color.BLACK);
        typedArray.recycle();
    }
    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
        int widthMode = MeasureSpec.getMode(widthMeasureSpec);
        int widthSize = MeasureSpec.getSize(widthMeasureSpec);
        int heightMode = MeasureSpec.getMode(heightMeasureSpec);
        int heightSize = MeasureSpec.getSize(heightMeasureSpec);
        switch (widthMode) {
            case MeasureSpec.EXACTLY: break;
            case MeasureSpec.AT_MOST: break;
            case MeasureSpec.UNSPECIFIED: break;
        }
    }
    @Override
    protected void onLayout(boolean changed, int left, int top, int right, int bottom) {
        super.onLayout(changed, left, top, right, bottom);
        Log.e(TAG, "onLayout");
    }
    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        // FILL填充, STROKE描边,FILL_AND_STROKE填充和描边
        mPaint.setStyle(Paint.Style.FILL_AND_STROKE);
        int width = getWidth();
        int height = getHeight();
        //Log.e(TAG, "onDraw---->" + width + "*" + height);
        float radius = width/4;

        mPaint.setColor(mShadowColor);
        canvas.drawCircle(width/2+mDx,+width/2+mDy,radius,mPaint);

        mPaint.setColor(Color.CYAN);
        canvas.drawCircle(width/2, width/2, radius, mPaint);
        mPaint.setColor(Color.BLUE);
        oval.set(width/2 - radius, width/2 - radius, width/2
                + radius, width/2 + radius);//用于定义的圆弧的形状和大小的界限
        canvas.drawArc(oval, 270, 120, true, mPaint);  //根据进度画圆弧
        Bitmap bitmap=BitmapFactory.decodeResource(mContext.getResources(),mBitmapID);
        canvas.drawBitmap(bitmap,width/2,width/2,mPaint);
    }
}
```