**View绘制2-onMeasure**

在自定义View的绘制过程中，重写onMeasure，onLayout,onDraw三个函数实现了View的外观形象，加上onTouchEvent等等函数实现的重载视图行为，构建出一个完整的自定义View体系。  
在Android体系中，以on来头的onXXX函数，多以在Activity，Service，View中出现，一般都是使用了设计模式里面的模板设计模式。定义好一套模板流程，然后通过重写模板方法实现自定义效果。

**作用**

1. onMeasure指定绘制View的大小
2. onLayout 指定绘制View的位置
3. onDraw 实现绘制过程  
   从系统源码看起

**View的onMeasure实现过程**

onMeasure( ) - 封装外部调用  
|  
setMeasuredDimension( ) -实现把测绘到的占用大小设置给View  
|  
getDefaultSize( )-比较Min大小和测绘大小做出抉择  
|  
getSuggestMinimumWidth( )-得到Min大小

分别的实现代码如下

protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

*//根据测绘大小的widthMeasureSpec和heightMeasureSpeac来确定子控件的大小*

setMeasuredDimension(getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec),

getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec));

}

具体实现方法封装在setMeasuredDimension()中

protected final void setMeasuredDimension(int measuredWidth, int measuredHeight) {

*//一系列程序健壮性判断 代码省略....*

mMeasuredWidth = measuredWidth;

mMeasuredHeight = measuredHeight;

mPrivateFlags |= PFLAG\_MEASURED\_DIMENSION\_SET;

}

public static int getDefaultSize(int size, int measureSpec) {

*//size默认大小*

int result = size;

int specMode = MeasureSpec.getMode(measureSpec);

int specSize = MeasureSpec.getSize(measureSpec);

switch (specMode) {

*//如果指定大小为不确定就使用默认大小*

case MeasureSpec.UNSPECIFIED:

result = size;

break;

*//否则使用测绘大小*

case MeasureSpec.AT\_MOST:

case MeasureSpec.EXACTLY:

result = specSize;

break;

}

return result;

}

getDefaultSize()方法返回MeasureSpec中的specSize，这个specSize就是View的测量大小，因为View的最终大小是在layout()中确定的，但是specSize的大小几乎所有时候都是和layout()中确定的最终大小相等

protected int getSuggestedMinimumWidth() {

*//mMinWidth可以通过xml布局设置android:minSize指定，也可以通过View.SetMinSize指定*

return (mBackground == null) ? mMinWidth : max(mMinWidth, mBackground.getMinimumWidth());

}

getSuggestedMinimumWidth()方法，android:minWidth如果有设置的话就设置宽度为这个值，但是还存在一种情况就是设置了Background的情况，这种情况下需要比较Background和minWidth的大小。  
上面就是通过widthMeasureSpec 和 heightMeasureSpec设置占用空间大小的过程，追本溯源widthMeasureSpec和heightMeasureSpec有是从何而来呢？

**MeasureSpec是什么**

测量规格，MeasureSpec有一个32位的int数表示，作用

1. 包含父布局对子布局View的测量要求
2. 包含测量模式和测量数据
3. 可以表示宽、高  
   1和3都好理解，不好理解的是第2点，查看MeasureSpec的源码

public static class MeasureSpec {

private static final int MODE\_SHIFT = 30;

/\*\* 工具位 \*/

private static final int MODE\_MASK = 0x3 << MODE\_SHIFT;

/\*\* 不确定模式 \*/

public static final int UNSPECIFIED = 0 << MODE\_SHIFT;

/\*\* 精确模式 \*/

public static final int EXACTLY = 1 << MODE\_SHIFT;

/\*\* 最大模式 \*/

public static final int AT\_MOST = 2 << MODE\_SHIFT;

/\*\*

\* 获取测量模式

\*/

public static int getMode(int measureSpec) {

return (measureSpec & MODE\_MASK);

}

/\*\*

\* 获取测量数据

\*/

public static int getSize(int measureSpec) {

return (measureSpec & ~MODE\_MASK);

}

/\*\*

\* 生成器

\*/

public static int makeMeasureSpec(int size, int mode) {

if (sUseBrokenMakeMeasureSpec) {

return size + mode;

} else {

return (size & ~MODE\_MASK) | (mode & MODE\_MASK);

}

}

static int adjust(int measureSpec, int delta) {

return makeMeasureSpec(getSize(measureSpec + delta),

getMode(measureSpec));

}

}

这里源码做了一些便于理解的删减。定义了一个标记为MODE\_MASK=3<<30;获取测量模式

public static int getMode(int measureSpec) {

return (measureSpec & MODE\_MASK);

}

和获取测量数据

public static int getSize(int measureSpec) {

return (measureSpec & ~MODE\_MASK);

}

可知这个int类型的数据32位，前2位表示测量模式，后30位表示测量数据  
其中

1. 00表示MeasureSpec.UNSPECIFIED 表示父布局对子布局不做任何限制，子控件想要多大就多大 这种模式一般不深究，一般系统用来对ListView和ScrollView这些控件使用。
2. 01表示 MeasureSpec.EXACTLY 表示精确控制 View的大小就是getSize（）返回的值
3. 10表示MeasureSpec.AT\_MOST 表示由子布局自己指配但是最大不能超过getSize( )的参考值

**从ViewGroup得到MeasureSpec的过程**

从measureChildWithMargins方法看起

protected void measureChildWithMargins(View child,

int parentWidthMeasureSpec, int widthUsed,

int parentHeightMeasureSpec, int heightUsed) {

final MarginLayoutParams lp = (MarginLayoutParams) child.getLayoutParams();

final int childWidthMeasureSpec = getChildMeasureSpec(parentWidthMeasureSpec,

mPaddingLeft + mPaddingRight + lp.leftMargin + lp.rightMargin

+ widthUsed, lp.width);

final int childHeightMeasureSpec = getChildMeasureSpec(parentHeightMeasureSpec,

mPaddingTop + mPaddingBottom + lp.topMargin + lp.bottomMargin

+ heightUsed, lp.height);

child.measure(childWidthMeasureSpec, childHeightMeasureSpec);

}

包含5个参数：  
子View，父WidthMeasureSpec、父HeightMeasureSpec、已经使用的宽度、已经使用的高度  
执行过程：  
1. 首先拿到LayoutParams  
2. 获取View的WidthMeasureSpec  
3. 获取View的HeightMeasureSpec  
4. 测绘child.measure(childWidthMeasureSpec, childHeightMeasureSpec);

public static int getChildMeasureSpec(int spec, int padding, int childDimension) {

int specMode = MeasureSpec.getMode(spec);

int specSize = MeasureSpec.getSize(spec);

int size = Math.max(0, specSize - padding);

int resultSize = 0;

int resultMode = 0;

switch (specMode) {

*// Parent has imposed an exact size on us*

case MeasureSpec.EXACTLY:

if (childDimension >= 0) {

resultSize = childDimension;

resultMode = MeasureSpec.EXACTLY;

} else if (childDimension == LayoutParams.MATCH\_PARENT) {

*// Child wants to be our size. So be it.*

resultSize = size;

resultMode = MeasureSpec.EXACTLY;

} else if (childDimension == LayoutParams.WRAP\_CONTENT) {

*// Child wants to determine its own size. It can't be*

*// bigger than us.*

resultSize = size;

resultMode = MeasureSpec.AT\_MOST;

}

break;

*// Parent has imposed a maximum size on us*

case MeasureSpec.AT\_MOST:

if (childDimension >= 0) {

*// Child wants a specific size... so be it*

resultSize = childDimension;

resultMode = MeasureSpec.EXACTLY;

} else if (childDimension == LayoutParams.MATCH\_PARENT) {

*// Child wants to be our size, but our size is not fixed.*

*// Constrain child to not be bigger than us.*

resultSize = size;

resultMode = MeasureSpec.AT\_MOST;

} else if (childDimension == LayoutParams.WRAP\_CONTENT) {

*// Child wants to determine its own size. It can't be*

*// bigger than us.*

resultSize = size;

resultMode = MeasureSpec.AT\_MOST;

}

break;

*// Parent asked to see how big we want to be*

case MeasureSpec.UNSPECIFIED:

if (childDimension >= 0) {

*// Child wants a specific size... let him have it*

resultSize = childDimension;

resultMode = MeasureSpec.EXACTLY;

} else if (childDimension == LayoutParams.MATCH\_PARENT) {

*// Child wants to be our size... find out how big it should*

*// be*

resultSize = 0;

resultMode = MeasureSpec.UNSPECIFIED;

} else if (childDimension == LayoutParams.WRAP\_CONTENT) {

*// Child wants to determine its own size.... find out how*

*// big it should be*

resultSize = 0;

resultMode = MeasureSpec.UNSPECIFIED;

}

break;

}

return MeasureSpec.makeMeasureSpec(resultSize, resultMode);

}

执行过程  
1. 获取父specMode 和 specSize  
2. 获取水平（垂直）方向最大可用空间 size  
3. 通过specMode 和 childDimension（view的空间大小）来确定子View的MeasureSpec  
这里我们知道，子View的控件的占用大小是由子View和他的ViewGroup共同决定的，具体关系可以参考下表：

| **parentSpecMode & childViewSize** | **EXACTLY** | **AT\_MOST** | **UNSPECIFIED** |
| --- | --- | --- | --- |
| 确定的值，如：100dp | EXACTILY & childSize | AT\_MOST& childSize | AT\_MOST& childSize |
| match\_parent | EXACTILY & parentLeftSize | AT\_MOST& parentLeftSize | UNSPECIFIED & 0 |
| wrap\_content | AT\_MOST& parentLeftSize | AT\_MOST& parentLeftSize | UNSPECIFIED & 0 |

通过表可以清除的发现，只要子View是具体的值那么不管父ViewGrounp的测量模式他都是EXACTILY + 子View具体的值

**自定义View我们还需要做什么**

| **parentSpecMode & childViewSize** | **EXACTLY** | **AT\_MOST** | **UNSPECIFIED** |
| --- | --- | --- | --- |
| 确定的值，如：100dp | EXACTILY & childSize | AT\_MOST& childSize | AT\_MOST& childSize |
| match\_parent | EXACTILY & parentLeftSize | AT\_MOST& parentLeftSize | UNSPECIFIED & 0 |
| wrap\_content | ***AT\_MOST& parentLeftSize*** | ***AT\_MOST& parentLeftSize*** | UNSPECIFIED & 0 |

根据源码得到的表中加粗加斜体这两项，逻辑上存在问题，比如一个控件如果指定他的高度为android:height=wrap\_content那么就应该由他自己来设置高度的大小，而不是去匹配他父ViewGroup的大小。设置为 AT\_MOST& parentLeftSizewrap\_content和match\_parent没有区别，实际上在系统自定义控件如TextView ImageView的onMeasure方法也是改写过的，wrap\_content模式下让子View自己去指配自己的大小。重写onMeasure()实现代码：

protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

*//遵循模板方法，其他逻辑不变*

super.onMeasure(widthMeasureSpec , heightMeasureSpec);

int widthSpecMode = MeasureSpec.getMode(widthMeasureSpec);

int widthSpceSize = MeasureSpec.getSize(widthMeasureSpec);

int heightSpecMode=MeasureSpec.getMode(heightMeasureSpec);

int heightSpceSize=MeasureSpec.getSize(heightMeasureSpec);

*//判断如果是At\_Most情况下做相应的处理 if(widthSpecMode==MeasureSpec.AT\_MOST&&heightSpecMode==MeasureSpec.AT\_MOST){*

setMeasuredDimension(mWidth, mHeight);

}else if(widthSpecMode==MeasureSpec.AT\_MOST){

setMeasuredDimension(mWidth, heightSpceSize);

}else if(heightSpecMode==MeasureSpec.AT\_MOST){

setMeasuredDimension(widthSpceSize, mHeight);

}

}

在上面的代码中，只需在wrap\_content的时候给mWidth mHeight设置一个默认的高度即可，至于具体的值需要具体分析。

**逻辑上还存在不对的地方**

| **parentSpecMode & childViewSize** | **EXACTLY** | **AT\_MOST** | **UNSPECIFIED** |
| --- | --- | --- | --- |
| 确定的值，如：100dp | EXACTILY & childSize | AT\_MOST& childSize | AT\_MOST& childSize |
| match\_parent | EXACTILY & parentLeftSize | ***AT\_MOST& parentLeftSize*** | UNSPECIFIED & 0 |
| wrap\_content | AT\_MOST& parentLeftSize | AT\_MOST& parentLeftSize | UNSPECIFIED & 0 |

分析如果子View是math\_parent父ViewGroup为AT\_MOST的情况是否存在。

1. ViewGroup是一个确定的值，那么根据第一横排的信息他一定是EXACTILY类型，错误
2. ViewGroup是wrap\_content，那么子View是match\_parent，子View的大小取决于父ViewGroup的大小，但是父ViewGroup的大小又是根据他包括的内容确定，互相持有对方的依赖，相当于操作系统的**死锁**，所以这种情况不复存在。
3. ViewGroup是match\_content，那么ViewGroup的父布局也一定是match\_content的（如果是wrap\_content和确定的值可以参考上面1,2分析），由此类推ViewGroup的父布局的父布局也一定是match\_content，一次类推….，但是到最终肯定会出现一个根布局不是match\_content类型，因为手机屏幕是客观存在且有大小的一个事物，所以ViewGroup是match\_content这种类型也不存在。  
   总之，没有任何一种情况可以让View是match\_parent类型而且View的父布局是AT\_MOST的。

**最后**

measure过程是View三大流程中最复杂的一个，在measure完成后通过getWidthMeasure() getHeightMeasure()方法可以获取到正确的宽高。但是在一些特殊情况下，系统需要多次measure才能确定最终的宽高，就种情况在onMeasure方法中拿到的测量宽高可能不准确。一个好的习惯是在onLayout()中获取View的测量宽高和最终宽高。  
如果有一个需求是在Activity一启动就去获取一个View的宽高。可能会想到在生命周期方法onCreate()/onStart()/onResume()中，但是measure和生命周期的方法并不同步，如果在特定的生命周期方法中获取而measure还没执行完拿到的值很可能是0.  
通过onWindowFocusChanged()在View绘制完成后，焦点肯定会改变，同时如果频繁进行onResume和onPause的话onWindowFocusChanged()也会执行  
代码：

@Override

public void onWindowFocusChanged(boolean hasFocus) {

super.onWindowFocusChanged(hasFocus);

if (hasFocus) {

int width = view.getMeasuredWidth();

int height = view.getMeasuredHeight();

}

}

在view绘制线程post一个消息在尾部，view绘制完成后会执行这个runnable

@Override

protected void onStart() {

super.onStart();

view.post(new Runnable() {

@Override

public void run() {

int width = view.getMeasuredWidth();

int height = view.getMeasuredHeight();

}

});

}

参考：《Android开发艺术探讨》-任玉刚

[**逆水当行舟**](https://blog.csdn.net/zivensonice)