



# 深入理解Android之View的绘制流程

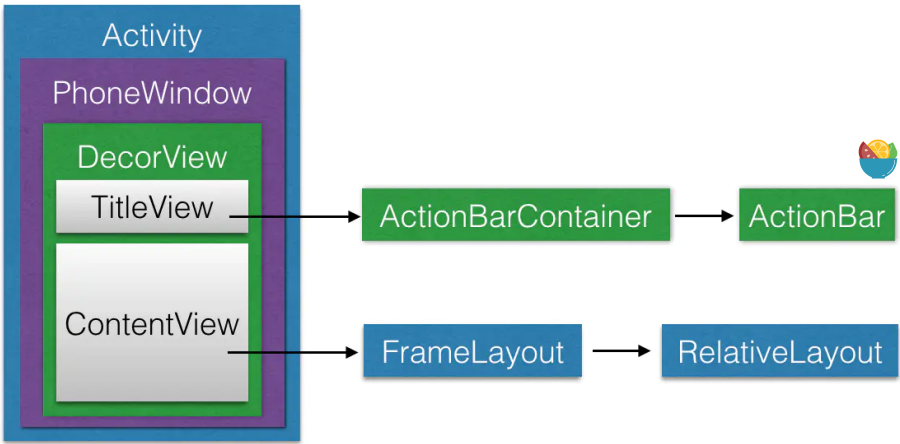


absfree 关注

7 2016.11.07 00:57:35 字数 4,962 阅读 62,715

## 概述

本篇文章会从源码（基于Android 6.0）角度分析Android中View的绘制流程，侧重于对整体流程的分析，对一些难以理解的点加以重点阐述，目的是把View绘制的整个流程把握好，而对于特定实现细节则可以日后对相应源码进行研读。  
在进行实际的分析之前，我们先来看下面这张图：



我们来对上图做出简单解释：DecorView是一个应用窗口的根容器，它本质上是一个FrameLayout。DecorView有唯一一个子View，它是一个垂直LinearLayout，包含两个子元素，一个是TitleView（ActionBar的容器），另一个是ContentView（窗口内容的容器）。关于ContentView，它是一个FrameLayout（android.R.id.content），我们平常用的setContentView就是设置它的子View。上图还表达了每个Activity都与一个Window（具体来说是PhoneWindow）相关联，用户界面则由Window所承载。

## Window

Window即窗口，这个概念在Android Framework中的实现为android.view.Window这个抽象类，这个抽象类是对Android系统中的窗口的抽象。在介绍这个类之前，我们先来看看究竟什么是窗口呢？

实际上，窗口是一个宏观的思想，它是屏幕上用于绘制各种UI元素及响应用户输入事件的一个矩形区域。通常具备以下两个特点：

- 独立绘制，不与其它界面相互影响；
- 不会触发其它界面的输入事件；

在Android系统中，窗口是独占一个Surface实例的显示区域，每个窗口的Surface由WindowManagerService分配。我们可以把Surface看作一块画布，应用可以通过Canvas或OpenGL在其上面作画。画好之后，通过SurfaceFlinger将多块Surface按照特定的顺序（即Z-

### 推荐阅读

1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单



写下你的评论...

评论23

赞254

...

# 深入理解Android之View的绘制流程



absfree

关注

赞赏支持

来介绍一下android.view.Window这个抽象类。

这个抽象类包含了三个核心组件：

- WindowManager.LayoutParams: 窗口的布局参数；
- Callback: 窗口的回调接口，通常由Activity实现；
- ViewTree: 窗口所承载的控件树。

下面我们来看一下Android中Window的具体实现（也是唯一实现）——PhoneWindow。

## PhoneWindow

前面我们提到了，PhoneWindow这个类是Framework为我们提供的Android窗口的具体实现。我们平时调用setContentView()方法设置Activity的用户界面时，实际上就完成了对所关联的PhoneWindow的ViewTree的设置。我们还可以通过Activity类的requestWindowFeature()方法来定制Activity关联PhoneWindow的外观，这个方法实际上做的是把我们所请求的窗口外观特性存储到了PhoneWindow的mFeatures成员中，在窗口绘制阶段生成外观模板时，会根据mFeatures的值绘制特定外观。

## 从setContentView()说开去



在分析setContentView()方法前，我们需要明确：这个方法只是完成了Activity的ContentView的创建，而并没有执行View的绘制流程。

当我们自定义Activity继承自android.app.Activity时候，调用的setContentView()方法是Activity类的，源码如下：

```
1 public void setContentView(@LayoutRes int layoutResID) {
2     getWindow().setContentView(layoutResID);
3     ...
4 }
```

getWindow()方法会返回Activity所关联的PhoneWindow，也就是说，实际上调用了PhoneWindow的setContentView()方法，源码如下：

```
1 @Override
2 public void setContentView(int layoutResID) {
3     if (mContentParent == null) {
4         // mContentParent即为上面提到的ContentView的父容器，若为空则调用installDecor()生成
5         installDecor();
6     } else if (!hasFeature(FEATURE_CONTENT_TRANSITIONS)) {
7         // 具有FEATURE_CONTENT_TRANSITIONS特性表示开启了Transition
8         // mContentParent不为null，则移除decorView的所有子View
9         mContentParent.removeAllViews();
10    }
11    if (hasFeature(FEATURE_CONTENT_TRANSITIONS)) {
12        // 开启了Transition，做相应的处理，我们不讨论这种情况
13        // 感兴趣的同学可以参考源码
14        ...
15    } else {
16        // 一般会来到这里，调用mLayoutInflater.inflate()方法来填充布局
17        // 填充布局也就是把我们设置的ContentView加入到mContentParent中
18        mLayoutInflater.inflate(layoutResID, mContentParent);
19    }
20    ...
21    // cb即为该Window所关联的Activity
22    final Callback cb = getCallback();
23    if (cb != null && !isDestroyed()) {
24        // 调用onContentChanged()回调方法通知Activity窗口内容发生了改变
25        cb.onContentChanged();
26    }
27 }
```

### 推荐阅读

1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单



写下你的评论...

评论23

赞254

# 深入理解Android之View的绘制流程



absfree

关注

赞赏支持

在上面我们看到了，PhoneWindow的setContentView()方法中调用了LayoutInflater的inflate()方法来填充布局，这个方法的源码如下：

```
1 public View inflate(@LayoutRes int resource, @Nullable ViewGroup root) {
2     return inflate(resource, root, root != null);
3 }
4
5 public View inflate(@LayoutRes int resource, @Nullable ViewGroup root, boolean attachToRoot) {
6     final Resources res = getContext().getResources();
7     ...
8     final XmlPullParser parser = res.getLayout(resource);
9     try {
10         return inflate(parser, root, attachToRoot);
11     } finally {
12         parser.close();
13     }
14 }
```

在PhoneWindow的setContentView()方法中传入了decorView作为LayoutInflater.inflate()的root参数，我们可以看到，通过层层调用，最终调用的是inflate(XmlPullParser, ViewGroup, boolean)方法来填充布局。这个方法的源码如下：

```
1 public View inflate(XmlPullParser parser, @Nullable ViewGroup root, boolean attachToRoot) {
2     synchronized (mConstructorArgs) {
3         ...
4         final Context inflaterContext = mContext;
5         final AttributeSet attrs = Xml.asAttributeSet(parser);
6         Context lastContext = (Context) mConstructorArgs[0];
7         mConstructorArgs[0] = inflaterContext;
8
9         View result = root;
10
11         try {
12             // Look for the root node.
13             int type;
14             // 一直读取xml文件，直到遇到开始标记
15             while ((type = parser.next()) != XmlPullParser.START_TAG &&
16                 type != XmlPullParser.END_DOCUMENT) {
17                 // Empty
18             }
19             // 最先遇到的不是开始标记，报错
20             if (type != XmlPullParser.START_TAG) {
21                 throw new InflateException(parser.getPositionDescription()
22 + ": No start tag found!");
23             }
24
25             final String name = parser.getName();
26             ...
27             // 单独处理<merge>标签，不熟悉的同学请参考官方文档的说明
28             if (TAG_MERGE.equals(name)) {
29                 // 若包含<merge>标签，父容器（即root参数）不可为空且attachToRoot须为true，否则报错
30                 if (root == null || !attachToRoot) {
31                     throw new InflateException("<merge /> can be used only with a valid "
32 + "ViewGroup root and attachToRoot=true");
33                 }
34
35                 // 递归地填充布局
36                 rInflate(parser, root, inflaterContext, attrs, false);
37             } else {
38                 // temp为xml布局文件的根View
39                 final View temp = createViewFromTag(root, name, inflaterContext, attrs);
40                 ViewGroup.LayoutParams params = null;
41                 if (root != null) {
42                     ...
43                     // 获取父容器的布局参数 (LayoutParams)
44                     params = root.generateLayoutParams(attrs);
45                     if (!attachToRoot) {
46                         // 若attachToRoot参数为false，则我们只会将父容器的布局参数设置给根View
47                         temp.setLayoutParams(params);
48                     }
49                 }
50             }
51         }
52     }
53 }
```

## 推荐阅读

1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)

阅读 228

【Android源码系列】深入源码分析UI的绘制流程

阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？

阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸

阅读 129



猎头公司名单



absfree

关注

赞赏支持

## 深入理解Android之View的绘制流程

```
56         if (root != null && attachToRoot) {
57             // 若父容器不为空且attachToRoot为true, 则将父容器作为根View的父View包裹上来
58             root.addView(temp, params);
59         }
60
61         // 若root为空或是attachToRoot为false, 则以根View作为返回值
62         if (root == null || !attachToRoot) {
63             result = temp;
64         }
65     }
66
67 } catch (XmlPullParserException e) {
68     . . .
69 } catch (Exception e) {
70     . . .
71 } finally {
72     . . .
73 }
74 }
75 return result;
76 }
77 }
```

在上面的源码中, 首先对于布局文件中的<merge>标签进行单独处理, 调用rInflate()方法来递归填充布局。这个方法的源码如下:

```
1 void rInflate(XmlPullParser parser, View parent, Context context,
2   AttributeSet attrs, boolean finishInflate) throws XmlPullParserException, IOException {
3     // 获取当前标记的深度, 根标记的深度为0
4     final int depth = parser.getDepth();
5     int type;
6     while (((type = parser.next()) != XmlPullParser.END_TAG ||
7       parser.getDepth() > depth) && type != XmlPullParser.END_DOCUMENT) {
8         // 不是开始标记则继续下一次迭代
9         if (type != XmlPullParser.START_TAG) {
10             continue;
11         }
12         final String name = parser.getName();
13         // 对一些特殊标记做单独处理
14         if (TAG_REQUEST_FOCUS.equals(name)) {
15             parseRequestFocus(parser, parent);
16         } else if (TAG_TAG.equals(name)) {
17             parseViewTag(parser, parent, attrs);
18         } else if (TAG_INCLUDE.equals(name)) {
19             if (parser.getDepth() == 0) {
20                 throw new InflateException("<include /> cannot be the root element");
21             }
22             // 对<include>做处理
23             parseInclude(parser, context, parent, attrs);
24         } else if (TAG_MERGE.equals(name)) {
25             throw new InflateException("<merge /> must be the root element");
26         } else {
27             // 对一般标记的处理
28             final View view = createViewFromTag(parent, name, context, attrs);
29             final ViewGroup viewGroup = (ViewGroup) parent;
30             final ViewGroup.LayoutParams params=viewGroup.generateLayoutParams(attrs);
31             // 递归地加载子View
32             rInflateChildren(parser, view, attrs, true);
33             viewGroup.addView(view, params);
34         }
35     }
36
37     if (finishInflate) {
38         parent.onFinishInflate();
39     }
40 }
```

我们可以看到, 上面的inflate()和rInflate()方法中都调用了rInflateChildren()方法, 这个方法的源码如下:

```
1 final void rInflateChildren(XmlPullParser parser, View parent, AttributeSet attrs, boolean finishInflate) {
2     rInflate(parser, parent, parent.getContext(), attrs, finishInflate);
3 }
```

### 推荐阅读

1.Android 花费5年 自定义view面试题都在这 (5分钟入门到牛逼) 面...  
阅读 1,317

高级UI-自定义View(一)

阅读 228

【Android源码系列】深入源码分析UI的绘制流程

阅读 90

什么? 这么精髓的View的Measure流程源码全解析, 你确定不看看?

阅读 245

Android (23) ——测量与布局: 子控件确定, 计算父容器尺寸

阅读 129



猎头公司名单

深入理解Android之View的绘制流程

到这里，setContentView()的整体执行流程我们就分析完了，至此我们已经完成了Activity的ContentView的创建与设置工作。接下来，我们开始进入正题，分析View的绘制流程。

ViewRoot

在介绍View的绘制前，首先我们需要知道是谁负责执行View绘制的整个流程。实际上，View的绘制是由ViewRoot来负责的。每个应用程序窗口的decorView都有一个与之关联的ViewRoot对象，这种关联关系是由WindowManager来维护的。

那么decorView与ViewRoot的关联关系是在什么时候建立的呢？答案是Activity启动时，ActivityThread.handleResumeActivity()方法中建立了它们两者的关联关系。这里我们不具体分析它们建立关联的时机与方式，感兴趣的同学可以参考相关源码。下面我们直入主题，分析一下ViewRoot是如何完成View的绘制的。

View绘制的起点

当建立好了decorView与ViewRoot的关联后，ViewRoot类的requestLayout()方法会被调用，以完成应用程序用户界面的初次布局。实际被调用的是ViewRootImpl类的requestLayout()方法，这个方法的源码如下：



```
1  @Override
2  public void requestLayout() {
3      if (!mHandlingLayoutInLayoutRequest) {
4          // 检查发起布局请求的线程是否为主线程
5          checkThread();
6          mLayoutRequested = true;
7          scheduleTraversals();
8      }
9  }
```

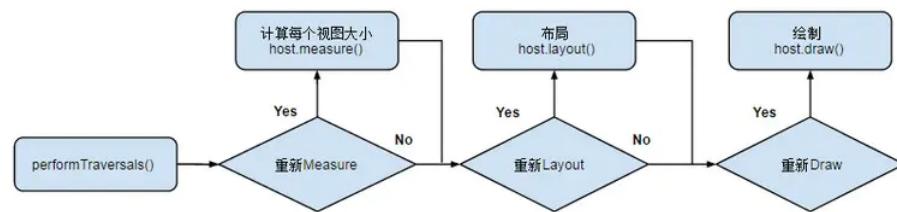
上面的方法中调用了scheduleTraversals()方法来调度一次完成的绘制流程，该方法会向主线程发送一个“遍历”消息，最终会导致ViewRootImpl的performTraversals()方法被调用。下面，我们以performTraversals()为起点，来分析View的整个绘制流程。

三个阶段

View的整个绘制流程可以分为以下三个阶段：

- measure: 判断是否需要重新计算View的大小，需要的话则计算；
- layout: 判断是否需要重新计算View的位置，需要的话则计算；
- draw: 判断是否需要重新绘制View，需要的话则重绘制。

这三个子阶段可以用下图来描述：



measure阶段

此阶段的目的是计算出控件树中的各个控件要显示其内容的话，需要多大尺寸。起点是

ViewRootImpl.performTraversals()方法，这个方法源码如下：

写下你的评论...

评论23

赞254



absfree

关注

赞赏支持

推荐阅读

1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单



深入理解Android之View的绘制流程



absfree

关注

赞赏支持

```
3 // 传入的desiredWindowWidth为窗口宽度
4 int childWidthMeasureSpec;
5 int childHeightMeasureSpec;
6 boolean windowSizeMayChange = false;
7 ...
8 boolean goodMeasure = false;
9
10 if (!goodMeasure) {
11     childWidthMeasureSpec = getRootMeasureSpec(desiredWindowWidth, lp.width);
12     childHeightMeasureSpec = getRootMeasureSpec(desiredWindowHeight, lp.height);
13     performMeasure(childWidthMeasureSpec, childHeightMeasureSpec);
14
15     if (mWidth != host.getMeasuredWidth() || mHeight != host.getMeasuredHeight()) {
16         windowSizeMayChange = true;
17     }
18 }
19 return windowSizeMayChange;
20 }
```

上面的代码中调用getRootMeasureSpec()方法来获取根MeasureSpec，这个根MeasureSpec代表了对decorView的宽高的约束信息。继续分析之前，我们先来简单地介绍下MeasureSpec的概念。

MeasureSpec是一个32位整数，由SpecMode和SpecSize两部分组成，其中，高2位为SpecMode，低30位为SpecSize。SpecMode为测量模式，SpecSize为相应测量模式下的测量尺寸。View（包括普通View和ViewGroup）的SpecMode由本View的LayoutParams结合父View的MeasureSpec生成。



SpecMode的取值可为以下三种：

- EXACTLY: 对子View提出了一个确切的建议尺寸（SpecSize）；
- AT\_MOST: 子View的大小不得超过SpecSize；
- UNSPECIFIED: 对子View的尺寸不作限制，通常用于系统内部。

传入performMeasure()方法的MeasureSpec的SpecMode为EXACTLY，SpecSize为窗口尺寸。

performMeasure()方法的源码如下：

```
1 private void performMeasure(int childWidthMeasureSpec, int childHeightMeasureSpec) {
2     ...
3     try {
4         mView.measure(childWidthMeasureSpec, childHeightMeasureSpec);
5     } finally {
6         ...
7     }
8 }
```

上面代码中的mView即为decorView，也就是说会转向对View.measure()方法的调用，这个方法的源码如下：

```
1 /**
2  * 调用这个方法来算出一个View应该为多大。参数为父View对其宽高的约束信息。
3  * 实际的测量工作在onMeasure()方法中进行
4  */
5 public final void measure(int widthMeasureSpec, int heightMeasureSpec) {
6     ...
7     // 判断是否需要重新布局
8
9     // 若mPrivateFlags中包含PFLAG_FORCE_LAYOUT标记，则强制重新布局
10    // 比如调用View.requestLayout()会在mPrivateFlags中加入此标记
11    final boolean forceLayout = (mPrivateFlags & PFLAG_FORCE_LAYOUT) == PFLAG_FORCE_LAYOUT;
12    final boolean specChanged = widthMeasureSpec != mOldWidthMeasureSpec
13        || heightMeasureSpec != mOldHeightMeasureSpec;
14    final boolean isSpecExactly = MeasureSpec.getMode(widthMeasureSpec) == MeasureSpec.EXACTLY
15        && MeasureSpec.getMode(heightMeasureSpec) == MeasureSpec.EXACTLY;
16    final boolean matchesSpecSize = getMeasuredWidth() == MeasureSpec.getSize(widthMeasureSpec)
17        && getMeasuredHeight() == MeasureSpec.getSize(heightMeasureSpec);
```

推荐阅读

1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单

写下你的评论...

评论23

赞254



absfree


关注

赞赏支持

## 深入理解Android之View的绘制流程

```
24 // 先尝试从缓存中获取, 若forceLayout为true或是缓存中不存在或是
25 // 忽略缓存, 则调用onMeasure()重新进行测量工作
26 int cacheIndex = forceLayout ? -1 : mMeasureCache.indexOfKey(key);
27 if (cacheIndex < 0 || sIgnoreMeasureCache) {
28     // measure ourselves, this should set the measured dimension flag back
29     onMeasure(widthMeasureSpec, heightMeasureSpec);
30     . . .
31 } else {
32     // 缓存命中, 直接从缓存中取值即可, 不必再测量
33     long value = mMeasureCache.valueAt(cacheIndex);
34     // Casting a long to int drops the high 32 bits, no mask needed
35     setMeasuredDimensionRaw((int) (value >> 32), (int) value);
36     . . .
37 }
38 . . .
39 }
40 mOldWidthMeasureSpec = widthMeasureSpec;
41 mOldHeightMeasureSpec = heightMeasureSpec;
42 mMeasureCache.put(key, ((long) mMeasuredWidth) << 32 |
43     (long) mMeasuredHeight & 0xffffffffL); // suppress sign extension
44 }
```

从measure()方法的源码中我们可以知道, 只有以下两种情况之一, 才会进行实际的测量工作:

- forceLayout为true: 这表示强制重新布局, 可以通过View.requestLayout()来实现;
- needsLayout为true, 这需要specChanged为true (表示本次传入的MeasureSpec与上次传入的不同), 并且以下三个条件之一成立: 
- sAlwaysRemeasureExactly为true: 该变量默认为false;
- isSpecExactly为false: 若父View对子View提出了精确的宽高约束, 则该变量为true, 否则为false
- matchesSpecSize为false: 表示父View的宽高尺寸要求与上次测量的结果不同

对于decorView来说, 实际执行测量工作的是FrameLayout的onMeasure()方法, 该方法的源码如下:

```
1 @Override
2 protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
3     int count = getChildCount();
4     . . .
5     int maxHeight = 0;
6     int maxWidth = 0;
7
8     int childState = 0;
9     for (int i = 0; i < count; i++) {
10         final View child = getChildAt(i);
11         if (mMeasureAllChildren || child.getVisibility() != GONE) {
12             measureChildWithMargins(child, widthMeasureSpec, 0, heightMeasureSpec, 0);
13             final LayoutParams lp = (LayoutParams) child.getLayoutParams();
14             maxWidth = Math.max(maxWidth,
15                 child.getMeasuredWidth() + lp.leftMargin + lp.rightMargin);
16             maxHeight = Math.max(maxHeight,
17                 child.getMeasuredHeight() + lp.topMargin + lp.bottomMargin);
18             childState = combineMeasuredStates(childState, child.getMeasuredState());
19         }
20     }
21 }
22
23 // Account for padding too
24 maxWidth += getPaddingLeftWithForeground() + getPaddingRightWithForeground();
25 maxHeight += getPaddingTopWithForeground() + getPaddingBottomWithForeground();
26
27 // Check against our minimum height and width
28 maxHeight = Math.max(maxHeight, getSuggestedMinimumHeight());
29 maxWidth = Math.max(maxWidth, getSuggestedMinimumWidth());
30
31 // Check against our foreground's minimum height and width
32 final Drawable drawable = getForeground();
33 if (drawable != null) {
34     . . .
35 }
```

### 推荐阅读

1.Android 花费5年 自定义view面试题都在这 (5分钟入门到牛逼) 面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么? 这么精髓的View的Measure流程源码全解析, 你确定不看看?  
阅读 245

Android (23) ——测量与布局: 子控件确定, 计算父容器尺寸  
阅读 129



猎头公司名单

写下你的评论...

评论23

赞254

# 深入理解Android之View的绘制流程



absfree

关注


赞赏支持

```
41         childState << MEASURED_HEIGHT_STATE_SHIFT));
42         . . .
43     }
```

FrameLayout是ViewGroup的子类，后者有一个View[]类型的成员变量mChildren，代表了其子View集合。通过getChildAt(i)能获取指定索引处的子View，通过getChildCount()可以获得子View的总数。

在上面的源码中，首先调用measureChildWithMargins()方法对所有子View进行了一遍测量，并计算出所有子View的最大宽度和最大高度。而后将得到的最大高度和宽度加上padding，这里的padding包括了父View的padding和前景区域的padding。然后会检查是否设置了最小宽高，并与其比较，将两者中较大的设为最终的最大宽高。最后，若设置了前景图像，我们还要检查前景图像的最小宽高。

经过了以上一系列步骤后，我们就得到了maxHeight和maxWidth的最终值，表示当前容器View用这个尺寸就能够正常显示其所有子View（同时考虑了padding和margin）。而后我们需要调用resolveSizeAndState()方法来结合传来的MeasureSpec来获取最终的测量宽高，并保存到mMeasuredWidth与mMeasuredHeight成员变量中。

从以上代码的执行流程中，我们可以看到，容器View通过measureChildWithMargins()方法对所有子View进行测量后，才能得到自身的测量结果。也就是说，对于ViewGroup及其子类说，要先完成子View的测量，再进行自身的测量（考虑进padding等）。

接下来我们来看下ViewGroup的measureChildWithMargins()方法的实现：

```
1  protected void measureChildWithMargins(View child,
2      int parentWidthMeasureSpec, int widthUsed,
3      int parentHeightMeasureSpec, int heightUsed) {
4      final MarginLayoutParams lp = (MarginLayoutParams) child.getLayoutParams();
5      final int childWidthMeasureSpec = getChildMeasureSpec(parentWidthMeasureSpec,
6          mPaddingLeft + mPaddingRight + lp.leftMargin + lp.rightMargin + widthUsed, lp.wi
7      final int childHeightMeasureSpec = getChildMeasureSpec(parentHeightMeasureSpec
8          mPaddingTop + mPaddingBottom + lp.topMargin + lp.bottomMargin + heightUsed, lp.h
9
10     child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
11
12 }
```

由以上代码我们可以知道，对于ViewGroup来说，它会调用child.measure()来完成子View的测量。传入ViewGroup的MeasureSpec是它的父View用于约束其测量的，那么ViewGroup本身也需要生成一个childMeasureSpec来限制它的子View的测量工作。这个childMeasureSpec就由getChildMeasureSpec()方法生成。接下来我们来分析这个方法：

```
1  public static int getChildMeasureSpec(int spec, int padding, int childDimension) {
2      // spec为父View的MeasureSpec
3      // padding为父View在相应方向的已用尺寸加上父View的padding和子View的margin
4      // childDimension为子View的LayoutParams的值
5      int specMode = MeasureSpec.getMode(spec);
6      int specSize = MeasureSpec.getSize(spec);
7
8      // 现在size的值为父View相应方向上的可用大小
9      int size = Math.max(0, specSize - padding);
10
11     int resultSize = 0;
12     int resultMode = 0;
13
14     switch (specMode) {
15         // Parent has imposed an exact size on us
16         case MeasureSpec.EXACTLY:
17             if (childDimension >= 0) {
18                 // 表示子View的LayoutParams指定了具体大小值 (xx dp)
19                 resultSize = childDimension;
20                 resultMode = MeasureSpec.EXACTLY;
21             } else if (childDimension == LayoutParams.MATCH_PARENT) {
```

## 推荐阅读

1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单



写下你的评论...

评论23

赞254



深入理解Android之View的绘制流程



absfree

关注

赞赏支持

```
27         resultSize = size;
28         resultMode = MeasureSpec.AT_MOST;
29     }
30     break;
31
32     // Parent has imposed a maximum size on us
33     case MeasureSpec.AT_MOST:
34         if (childDimension >= 0) {
35             // 子View指定了具体大小
36             resultSize = childDimension;
37             resultMode = MeasureSpec.EXACTLY;
38         } else if (childDimension == LayoutParams.MATCH_PARENT) {
39             // 子View想跟父View一样大，但是父View的大小未固定下来
40             // 所以指定约束子View不能比父View大
41             resultSize = size;
42             resultMode = MeasureSpec.AT_MOST;
43         } else if (childDimension == LayoutParams.WRAP_CONTENT) {
44             // 子View想要自己决定尺寸，但不能比父View大
45             resultSize = size;
46             resultMode = MeasureSpec.AT_MOST;
47         }
48         break;
49
50     . . .
51 }
52
53 //noinspection ResourceType
54 return MeasureSpec.makeMeasureSpec(resultSize, resultMode);
55 }
```



上面的方法展现了根据父View的MeasureSpec和子View的LayoutParams生成子View的MeasureSpec的过程，\*\*子View的LayoutParams表示了子View的期待大小\*\*。这个产生的MeasureSpec用于指导子View自身的测量结果的确定。

在上面的代码中，我们可以看到当ParentMeasureSpec的SpecMode为EXACTLY时，表示父View对子View指定了确切的宽高限制。此时根据子View的LayoutParams的不同，分以下三种情况：

- 具体大小（childDimension）：这种情况下令子View的SpecSize为childDimension，即子View在LayoutParams指定的具体大小值；令子View的SpecMode为EXACTLY，即这种情况下若该子View为容器View，它也有能力给其子View指定确切的宽高限制（子View只能在这个宽高范围内），若为普通View，它的最终测量大小就为childDimension。
- match\_parent：此时表示子View想和父View一样大。这种情况下得到的子View的SpecMode与上种情况相同，只不过SpecSize为size，即父View的剩余可用大小。
- wrap\_content：这表示了子View想自己决定自己的尺寸（根据其内容的大小动态决定）。这种情况下子View的确切测量大小只能在其本身的onMeasure()方法中计算得出，父View此时无从知晓。所以暂时将子View的SpecSize设为size（父View的剩余大小）；令子View的SpecMode为AT\_MOST，表示了若子View为ViewGroup，它没有能力给其子View指定确切的宽高限制，毕竟它本身的测量宽高还悬而未定。

当ParentMeasureSpec的SpecMode为AT\_MOST时，我们也可以根据子View的LayoutParams的不同来分三种情况讨论：

- 具体大小：这时令子View的SpecSize为childDimension，SpecMode为EXACTLY。
- match\_parent：表示子View想和父View一样大，故令子View的SpecSize为size，但是由于父View本身的测量宽高还无从确定，所以只是暂时令子View的测量结果为父View目前的可用大小。这时令子View的SpecMode为AT\_MOST。
- wrap\_content：表示子View想自己决定大小（根据其内容动态确定）。然而这时父View还无法确定其自身的测量宽高，所以暂时令子View的SpecSize为size，SpecMode为AT\_MOST。

由上面的分析我们可以得到一个简单的结论：当子View的测量结果能够确定时，子View的

推荐阅读

1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单



写下你的评论...

评论23

赞254

# 深入理解Android之View的绘制流程



absfree

关注

赞赏支持

在measureChildWithMargins()方法中，获取了知道子View测量的MeasureSpec后，接下来就要调用child.measure()方法，并把获取到的childMeasureSpec传入。这时便又会调用onMeasure()方法，若此时的子View为ViewGroup的子类，便会调用相应容器类的onMeasure()方法，其他容器Views的onMeasure()方法与FrameLayout的onMeasure()方法执行过程相似。

下面我们会回到FrameLayout的onMeasure()方法，当递归地执行完所有子View的测量工作后，会调用resolveSizeAndState()方法来根据之前的测量结果确定最终对FrameLayout的测量结果并存储起来。View类的resolveSizeAndState()方法的源码如下：

```
1 public static int resolveSizeAndState(int size, int measureSpec, int childMeasuredState) {
2     final int specMode = MeasureSpec.getMode(measureSpec);
3     final int specSize = MeasureSpec.getSize(measureSpec);
4     final int result;
5     switch (specMode) {
6         case MeasureSpec.AT_MOST:
7             if (specSize < size) {
8                 // 父View给定的最大尺寸小于完全显示内容所需尺寸
9                 // 则在测量结果上加上MEASURED_STATE_TOO_SMALL
10                result = specSize | MEASURED_STATE_TOO_SMALL;
11            } else {
12                result = size;
13            }
14            break;
15
16         case MeasureSpec.EXACTLY:
17             // 若specMode为EXACTLY，则不考虑size，result直接赋值为specSize
18             result = specSize;
19             break;
20
21         case MeasureSpec.UNSPECIFIED:
22             default:
23                 result = size;
24            }
25
26     return result | (childMeasuredState & MEASURED_STATE_MASK);
27 }
28 }
```



对于普通View，会调用View类的onMeasure()方法来进行实际的测量工作，该方法的源码如下：

```
1 protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
2     setMeasuredDimension(getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec),
3                           getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec));
4 }
```

对于普通View（非ViewGroup）来说，只需完成自身的测量工作即可。以上代码中通过setMeasuredDimension()方法设置测量的结果，具体来说是以getDefaultSize()方法的返回值来作为测量结果。getDefaultSize()方法的源码如下：

```
1 public static int getDefaultSize(int size, int measureSpec) {
2     int result = size;
3     int specMode = MeasureSpec.getMode(measureSpec);
4     int specSize = MeasureSpec.getSize(measureSpec);
5     switch (specMode) {
6         case MeasureSpec.UNSPECIFIED:
7             result = size;
8             break;
9         case MeasureSpec.AT_MOST:
10        case MeasureSpec.EXACTLY:
11            result = specSize;
12            break;
13    }
14    return result;
15 }
```

## 推荐阅读

- 1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317
- 高级UI-自定义View(一)  
阅读 228
- 【Android源码系列】深入源码分析UI的绘制流程  
阅读 90
- 什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245
- Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单



写下你的评论...

评论23

赞254

# 深入理解Android之View的绘制流程



absfree

关注

赞赏支持

wrap\_content (对应了AT\_MOST)这种情况进行处理，否则对自定义View指定wrap\_content就和match\_parent效果一样了。

## layout阶段

layout阶段的基本思想也是由根View开始，递归地完成整个控件树的布局（layout）工作。

### View.layout()

我们把对decorView的layout()方法的调用作为布局整个控件树的起点，实际上调用的是View类的layout()方法，源码如下：

```
1 public void layout(int l, int t, int r, int b) {
2     // l为本View左边缘与父View左边缘的距离
3     // t为本View上边缘与父View上边缘的距离
4     // r为本View右边缘与父View左边缘的距离
5     // b为本View下边缘与父View上边缘的距离
6     ...
7     boolean changed = isLayoutModeOptical(mParent) ? setOpticalFrame(l, t,
8     if (changed || (mPrivateFlags & PFLAG_LAYOUT_REQUIRED) == PFLAG_LAYOUT_REQUIRED) {
9         onLayout(changed, l, t, r, b);
10        ...
11    }
12    ...
13 }
14 }
```



这个方法会调用 setFrame() 方法来设置 View 的 mLeft、mTop、mRight 和 mBottom 四个参数，这四个参数描述了 View 相对其父 View 的位置（分别赋值为 l, t, r, b），在 setFrame() 方法中会判断 View 的位置是否发生了改变，若发生了改变，则需要对子 View 进行重新布局，对子 View 的局部是通过 onLayout() 方法实现了。由于普通 View（非 ViewGroup）不含子 View，所以 View 类的 onLayout() 方法为空。因此接下来，我们看看 ViewGroup 类的 onLayout() 方法的实现。

### ViewGroup.onLayout()

实际上 ViewGroup 类的 onLayout() 方法是 abstract，这是因为不同的布局管理器有着不同的布局方式。

这里我们以 decorView，也就是 FrameLayout 的 onLayout() 方法为例，分析 ViewGroup 的布局过程：

```
1 @Override
2 protected void onLayout(boolean changed, int left, int top, int right, int bottom) {
3     layoutChildren(left, top, right, bottom, false /* no force left gravity */);
4 }
5
6 void layoutChildren(int left, int top, int right, int bottom, boolean forceLeftGravity) {
7     final int count = getChildCount();
8     final int parentLeft = getPaddingLeftWithForeground();
9     final int parentRight = right - left - getPaddingRightWithForeground();
10    final int parentTop = getPaddingTopWithForeground();
11    final int parentBottom = bottom - top - getPaddingBottomWithForeground();
12
13    for (int i = 0; i < count; i++) {
14        final View child = getChildAt(i);
15        if (child.getVisibility() != GONE) {
16            final LayoutParams lp = (LayoutParams) child.getLayoutParams();
17            final int width = child.getMeasuredWidth();
18            final int height = child.getMeasuredHeight();
19            int childLeft;
20            int childTop;
21            int gravity = lp.gravity;
22
23            if (gravity == -1) {
```

#### 推荐阅读

1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单

写下你的评论...

评论23

赞254

深入理解Android之View的绘制流程



absfree

关注

赞赏支持

```
30
31     switch (absoluteGravity & Gravity.HORIZONTAL_GRAVITY_MASK) {
32         case Gravity.CENTER_HORIZONTAL:
33             childLeft = parentLeft + (parentRight - parentLeft - width) / 2 +
34             lp.leftMargin - lp.rightMargin;
35             break;
36
37         case Gravity.RIGHT:
38             if (!forceLeftGravity) {
39                 childLeft = parentRight - width - lp.rightMargin;
40                 break;
41             }
42
43         case Gravity.LEFT:
44         default:
45             childLeft = parentLeft + lp.leftMargin;
46
47     }
48
49     switch (verticalGravity) {
50         case Gravity.TOP:
51             childTop = parentTop + lp.topMargin;
52             break;
53
54         case Gravity.CENTER_VERTICAL:
55             childTop = parentTop + (parentBottom - parentTop - height) / 2 +
56             lp.topMargin - lp.bottomMargin;
57             break;
58
59         case Gravity.BOTTOM:
60             childTop = parentBottom - height - lp.bottomMargin;
61             break;
62
63         default:
64             childTop = parentTop + lp.topMargin;
65     }
66     child.layout(childLeft, childTop, childLeft + width, childTop + height);
67 }
68 }
69 }
```



推荐阅读

1.Android 花费5年 自定义view面试题都在这 （5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单

在上面的方法中，parentLeft表示当前View为其子View显示区域指定的一个左边界，也就是子View显示区域的左边缘到父View的左边缘的距离，parentRight、parentTop、parentBottom的含义同理。确定了子View的显示区域后，接下来，用一个for循环来完成子View的布局。在确保子View的可见性不为GONE的情况下才会对其进行布局。首先会获取子View的LayoutParams、layoutDirection等一系列参数。上面代码中的childLeft代表了最终子View的左边缘距父View左边缘的距离，childTop代表了子View的上边缘距父View的上边缘的距离。会根据子View的layout\_gravity的取值对childLeft和childTop做出不同的调整。最后会调用child.layout()方法对子View的位置参数进行设置，这时便转到了View.layout()方法的调用，若子View是容器View，则会递归地对其子View进行布局。

到这里，layout阶段的大致流程我们就分析完了，这个阶段主要就是根据上一阶段得到的View的测量宽高来确定View的最终显示位置。显然，经过了measure阶段和layout阶段，我们已经确定好了View的大小和位置，那么接下来就可以开始绘制View了。

draw阶段

对于本阶段的分析，我们以decorView.draw()作为分析的起点，也就是View.draw()方法，它的源码如下：

```
1 public void draw(Canvas canvas) {
2     ...
3     // 绘制背景，只有dirtyOpaque为false时才进行绘制，下同
4     int saveCount;
5     if (!dirtyOpaque) {
6         drawBackground(canvas);
7     }
```



写下你的评论...

评论23

赞254

# 深入理解Android之View的绘制流程

 absfree

关注

赞赏支持

```
13
14 // 绘制子View
15 dispatchDraw(canvas);
16
17 . . .
18 // 绘制滚动条等
19 onDrawForeground(canvas);
20
21 }
```

简单起见，在上面的代码中我们省略了实现滑动时渐变边框效果相关的逻辑。实际上，View类的onDraw()方法为空，因为每个View绘制自身的方式都不尽相同，对于decorView来说，由于它是容器View，所以它本身并没有什么要绘制的。dispatchDraw()方法用于绘制子View，显然普通View（非ViewGroup）并不能包含子View，所以View类中这个方法的实现为空。

ViewGroup类的dispatchDraw()方法中会依次调用drawChild()方法来绘制子View，drawChild()方法的源码如下：

```
1 protected boolean drawChild(Canvas canvas, View child, long drawingTime) {
2     return child.draw(canvas, this, drawingTime);
3 }
```

这个方法调用了View.draw(Canvas, ViewGroup, long)方法来对子View进行绘制。在draw(Canvas, ViewGroup, long)方法中，首先对canvas进行了一系列变换，以变换到将要被绘制的View的坐标系下。完成对canvas的变换后，便会调用View.draw(Canvas)方法进行实际的绘制工作，此时传入的canvas为经过变换的，在将被绘制View的坐标系下的canvas。

进入到View.draw(Canvas)方法后，会向之前介绍的一样，执行以下几步：

- 绘制背景;
- 通过onDraw()绘制自身内容;
- 通过dispatchDraw()绘制子View;
- 绘制滚动条

至此，整个View的绘制流程我们就分析完了。若文中有叙述不清晰或是不准确的地方，希望大家能够指出，谢谢大家：)

## 参考资料

- 《深入理解Android（卷三）》
- 《Android开发艺术探索》
- [公共技术点之View的绘制流程](#)

推荐阅读

1.Android 花费5年 自定义view面试题都在这（5分钟入门到牛逼）面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android（23）——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



长按或扫描二维码关注我们，让您利用每天等地铁的时间就能学会怎样写出优质app。



深入理解Android之View的绘制流程



absfree

关注

赞赏支持



254人点赞 >



Android通俗说



更多精彩内容，就在简书APP



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



absfree 把玩代码十三载

总资产16 (约0.80元) 共写了3.7W字 获得1,235个赞 共759个粉丝



关注

上海工位出租了，创业者一起办公，沟通起来更方便！



猎头公司名单

被以下专题收入，发现更多相似内容



Android



Android



面试



Android...



面试汇总



Android...



Android

展开更多 >

推荐阅读

更多精彩内容 >

Android View的测量、布局、绘制流程源码分析及自定义View...

在Android知识体系中，Android系统提供了一个GUI库，里面有很多原生控件，但是很多时候我们并不满足于系...

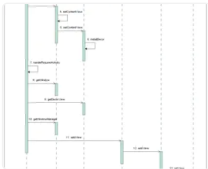


ForeverCy

阅读 6,027

评论 7

赞 52



自定义View系列教程02--onMeasure源码详尽分析

引言 本章内容较多，先养养眼 大家知道，自定义View有三个重要的步骤：measure, layout, draw。而...



SnowDragonYY

阅读 1,324

评论 1

赞 11



写下你的评论...

评论23

赞254

深入理解Android之View的绘制流程

absfree


关注

赞赏支持



View绘制流程及源码解析(二)——onMeasure()流程分析

接着上一篇View绘制流程及源码解析(一)——performTraversals()源码分析，这一篇我们来具体看看...

Geeks\_Liu

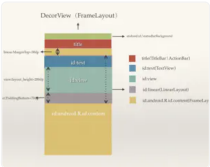
阅读 1,375 评论 1 赞 8

EXACTLY	AT_MOST
uiMode: EXACTLY	resultMode: EXACTLY
uiSize: childSize	resultSize: childSize
uiMode: EXACTLY	resultMode: AT_MOST
uiSize: parentSize	resultSize: parentSize
uiMode: AT_MOST	resultMode: AT_MOST
uiSize: parentSize	resultSize: parentSize

Android View的绘制流程

View的绘制和事件处理是两个重要的主题，上一篇《图解 Android事件分发机制》已经把事件的分发机制讲得比较详细...

Kelin

阅读 105,744 评论 95 赞 787

Android视图绘制流程完全解析，带你一步步深入了解View(二)

在上一篇文章中，我带着大家一起剖析了一下LayoutInflater的工作原理，可以算是对View进行深入了...

御风之

阅读 321 评论 0 赞 1

推荐阅读

1.Android 花费5年 自定义view面试题都在这 (5分钟入门到牛逼) 面...  
阅读 1,317

高级UI-自定义View(一)  
阅读 228

【Android源码系列】深入源码分析UI的绘制流程  
阅读 90

什么？这么精髓的View的Measure流程源码全解析，你确定不看看？  
阅读 245

Android (23) ——测量与布局：子控件确定，计算父容器尺寸  
阅读 129



猎头公司名单