Android View框架（系统）

分为两个部分，第一部分讲一下android framework层下View的工作原理，第二部分描述下自定义View的方法。

*注：源码均基于Android5.0代码*

# **零、引言：**

View

ViewGroup

Absolute

Layout

Linear

Layout

Relative

Layout

Frame

Layout

AdapterView(T

Extends Adapter)

TextView

ImageView

AbsList

View

GridView

ListView

AbsSpanner

Gallery

Android,Framework层的View类的继承关系如上图

# **一、View的生成：LayoutInflater**

## **1. LayoutInflater实例化：**

View的实例化可以java代码中添加，也可以xml中声明setContentView来解析；还有一种方法动态初始化view，就是LayoutInflater（例如onCreateView函数）。LayoutInflater实例化方式有两种：

LayoutInflater layoutInflater = LayoutInflater.from(context);

LayoutInflater layoutInflater = (LayoutInflater)context.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);

第一种方式是第二种方式的封装，简化实现方式。

## **2. LayoutInflater使用:**

例子：

**父parent：**

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/main\_layout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" >

</LinearLayout>

**子View：**

<Button xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Button" >

</Button>

**将子view加入父parent：**

View buttonLayout = layoutInflater.inflate(R.layout.button\_layout, null);

mainLayout.addView(buttonLayout);

LayoutInflater通过inflate（）方法，使用Android提供的pull解析方式来解析布局文件，通过循环调用递归将View树遍历解析，最后将rootView返回。完成了view的加载。

**做一些改动：**

子view：

<Button xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="80dp"

android:layout\_height="30dp"

android:text="Button" >

</Button>

其他不变，重新编译后，发现button大小并没有改变，layout\_width和layout\_height无论设置为多大都不变。

**解释：**

所以inflater方法有时会丢失layout的外层params信息。

LayoutInflater.java源码发现inflate的最终实现有三个参数：

public View inflate(int resource, ViewGroup root, boolean attachToRoot)

从代码注释可以看到：root作为resource的parent，如果root!=null && attachToRoot==true，

那么返回的View就是root，resource被add到root中；如果root == null,&& attachToRoot == false,那么返回view是resource，且resource的LayoutParams不被保留；如果root!=null&& attachToRoot ==null 那么返回source并且他的outParams被保留。

**注释：**

Activity的onCreate方法中setContentView（resId），外层也没有parent，为什么可以设置最外层view的大小？这主要是因为，在setContentView()方法中，Android会自动在布局文件的最外层再嵌套一个FrameLayout，所以layout\_width和layout\_height属性会有效果。

# **二、View的绘制流程**

了解其绘制原理，对于我们控件定制、扩展，bug的解决都有很大帮助。每一个视图的绘制过程都必须经历三个最主要的阶段，即onMeasure()、onLayout()和onDraw()。绘制过程按照**确定大小----确定位置----绘制** 的过程进行。下面逐一介绍：

## **onMeasure()**

ViewRoot的performTraversals()方法会根据View树遍历所有View，依次调用所有View的三个方法，直到绘制完成。

**public** **final** **void** [measure](http://androidxref.com/4.4_r1/s?refs=measure&project=frameworks" \t "_blank)(**int** [widthMeasureSpec](http://androidxref.com/4.4_r1/s?refs=widthMeasureSpec&project=frameworks" \t "_blank), **int** [heightMeasureSpec](http://androidxref.com/4.4_r1/s?refs=heightMeasureSpec&project=frameworks" \t "_blank))｛  
｝

传入的两个参数：widthMeasureSpec和heightMeasureSpec，是由specSize和specMode组成，specMode分为AT\_MOST, EXALTLY, UNSPECIFIED三种，specSize就是大小。为view的大小确定了约束条件。一般这两个参数是由parentView传入，也就是说view的大小是由parentView部分决定的。

**决定自身的大小：**

看一下View.java的measure方法（**final型**）：

onMeasure是核心，ViewGroup及其子类都继承了onMeasure：

ViewGroup除了确定自己的width和height之外，还要测量子view的大小:

**测量子View大小：**

protected void measureChildren(int widthMeasureSpec, int heightMeasureSpec) {

final int size = mChildrenCount;

final View[] children = mChildren;

for (int i = 0; i < size; ++i) {

final View child = children[i];

if ((child.mViewFlags & VISIBILITY\_MASK) != GONE) {

measureChild(child, widthMeasureSpec, heightMeasureSpec);

}

}

}

**举例**：（Demo）

public class MyView extends View {

......

@Override

protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

setMeasuredDimension(200, 200);

}

}

getMeasureWidth()获取到的mMeasuredWidth在setMeasuredDimension确定之前为均为0

**由此可见，视图大小的控制是由父视图、布局文件、以及子视图本身共同完成的，父视图会提供给子视图参考的大小，而开发人员可以在XML文件中指定视图的大小，然后视图本身会对最终的大小进行拍板。好的视图设计者会根据子视图的传入的measureSpec设置合适的布局大小，尊重程序员的意图。**

**Measure过程的本质是把视图布局时使用的“相对值”转换为具体值的过程。把WRAP\_CONTENT和MATCH\_PARENT转换为确定值。如果Framework没有相对值，那么就不必有measure这个过程了。**

举例：

CustomedFlowItemViewGroup.java onMeasure

## **2.onLayout()**

ViewRoot的performTraversals()方法会在measure遍历结束后继续执行，并调用View的layout()方法。

在layout()方法中，首先会调用setFrame()方法来判断视图的大小是否发生过变化，以确定有没有必要对当前的视图进行重绘。调用onLayout。实际上，ViewGroup和View的onLayout方法都是空，需要重写。可以看到LinearLayout,RelativeLayout等都是重写了onLayout。

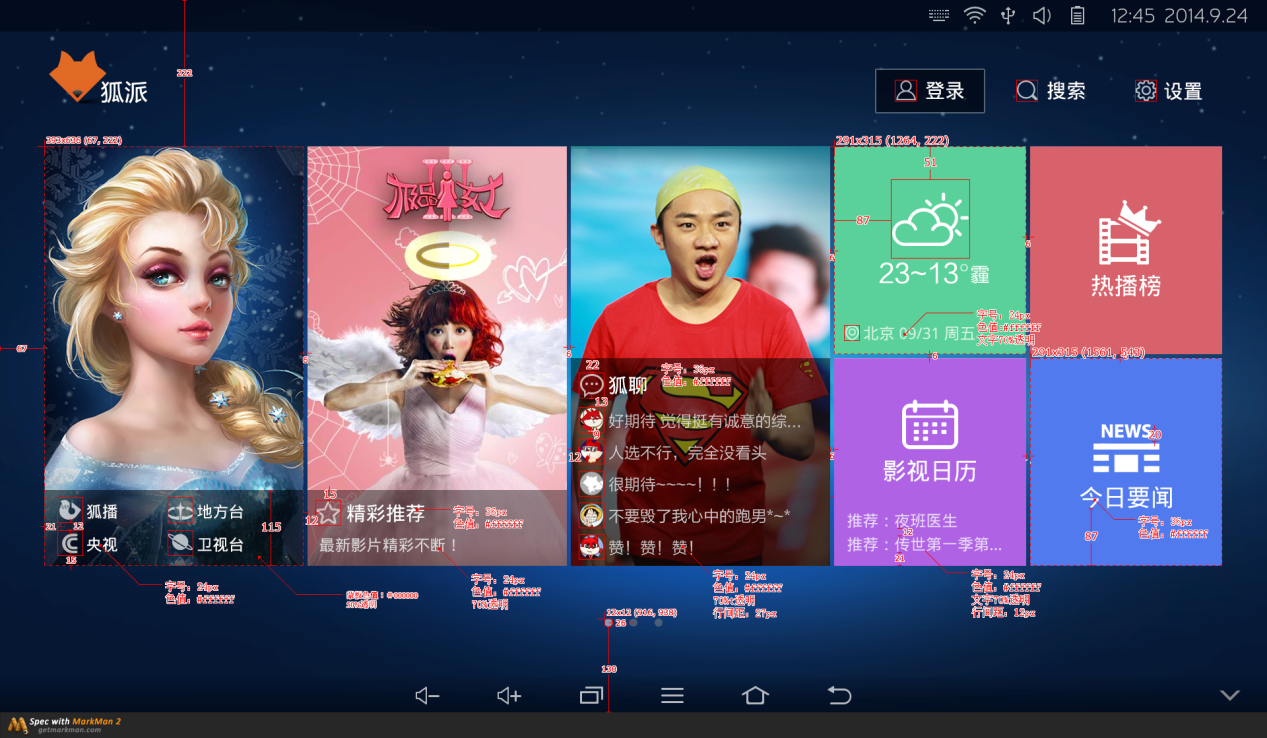
protected void onLayout(boolean changed, int left, int top, int right, int bottom) {

}

以UniversalCellLayout，CutomedFlowItmeViewGroup为例，说明一下onLayout的过程。

onLayout过程是控件对其内部子View布局的核心，直接影响到子View的排列方式。

**注释：**  
getWidth方法与getMeasuredWidth方法的区别，前面提到过，getMeasuredWidth必须在measure之后获取，getWidth需要在layout之后获取，视图的右边坐标减去左边坐标得到的。大部分情况下，二者是相同的。是因为布局设计者的编码习惯较好，按照（mLeft, mTop, mLeft + getMeasuredWidth , mTop + getMeasuredHeight）来布局。



## **3.onDraw（）**

首先，onDraw和DispatchDraw函数中不宜做复杂的算法和操作，会导致性能的下降。

View.java中Draw()函数：  
第一步：mBGDrawable绘制  
第二步：视图内容绘制，调用onDraw（）绘制自己，空实现  
第三步：disPatchDraw（）绘制子view，空实现  
第四步：滚动条scroll绘制  
**综上**，View是不会帮我们绘制内容部分的，因此需要每个视图根据想要展示的内容来自行绘制。如果你去观察TextView、ImageView等类的源码，你会发现它们都有重写onDraw()这个方法。  
绘制的过程主要是Cavas Paint等的运用，Cavas的裁剪移动决定了绘制的位置，Paint决定了绘制的内容。

更多内容可以参考下面的自定义view部分。

# **三、View的视图状态及重绘**

## **1.View的状态：**

View有多种状态(十几种状态)，一般常用状态：pressed，focused，selected，enabled，window\_focused 这些状态不展开讲，只讲分析方法。  
1）enable ：disable的view无法相应touch事件  
2）window\_focused: 当前视图是否处于正在交互的窗口中，这个值由系统自动决定，应用程序不能进行改变  
3) selected: 一般的基本控件如果不设置setFocusable（true），那么遥控器控制view的状态变化会直接影响selected状态。Focus和selected二者属于互斥的情况。  
4) focused: 见 framework对focus的移动判断  
5) pressed: ItemViewUnit 关于pressed的状态修改

## **2.view的重绘：**

调用视图的setVisibility()、setEnabled()、setSelected()等方法时都会导致视图重绘，当然如果要主动重绘，还可以调用invalidate,实际上上述三个方法也是间接调用invalidate实现的。View的源码中会有数个invalidate()方法的重载和一个invalidateDrawable()方法，最终实现是一致的。我们看下invalidate做了什么。

Android5.0 framework/base/core/java/android/view/View.java

invalidateInternal() –- p.invalidateChild(this, damage);

invalidateChild -- parent.invalidateChildInParent(location, dirty)

invalidateChildInParent – scheduleTraversals()

mTraversalRunnable – doTraversal() – performTraversals()

重新开始viewRoot的遍历更新流程，由于没有重新测量或大小变化的标志位，measure和layout都不会走，draw会走。requestLayout会调用measure和layout

# **四、自定义View**

## **1.自绘控件**

重写onDraw函数，适用于不需要更改大小位置，内容的更改不太复杂的情况，这样性能最优。复杂的实现可以参考TextView， ImageView的实现。  
这里有一个比较简单的实现：

protected void onDraw(Canvas canvas) {

super.onDraw(canvas);

mPaint.setColor(Color.BLUE);

canvas.drawRect(0, 0, getWidth(), getHeight(), mPaint);

mPaint.setColor(Color.YELLOW);

mPaint.setTextSize(30);

String text = String.valueOf(mCount);

mPaint.getTextBounds(text, 0, text.length(), mBounds);

float textWidth = mBounds.width();

float textHeight = mBounds.height();

canvas.drawText(text, getWidth() / 2 - textWidth / 2, getHeight() / 2

+ textHeight / 2, mPaint);

}

代码实现了简单的文字居中TextView功能

FoxpadLauncher中ItemViewUnit.java的实现：

**注意：自定义的View在使用的时候一定要写出完整的包名，不然系统将无法找到这个View**

## **2.组合控件**

利用android原生控件的组合，生成标准化的控件，并没有改变其实现方式。标题栏就是个很常见的组合控件，在SohuLauncher中，ActivityTopLayout是标准的实现。max3.0版本的视觉框架也是如此实现。  
这类的控件设计需要考虑，控件的扩展性和易用性。  
1）**扩展性**： 是否可扩展（外部layout最好使用扩展性强的layout）  
2）**易用性**：构造方便，足够的接口设置属性

## **3.继承(改造)控件**

**继承**：SmoothScrollView

**改造**：SmoothListView, SmoothGridView, SmoothViewPager, ListGallery,  
阅读控件思路：  
1）从交互入手（Touch事件，key事件）  
2）从初始化入手（如何Adapter取数据，onMeasure onLayout）  
改造控件思路：  
1）阅读的基础上，重点理解原控件的protected public函数，都是可以定制的部分，  
2）从以上可定制的部分，从需求、性能、可行性（别的控件实现方案）角度选择改写方案

更多参考：Trinea github(囊括所有主流自定义控件)

https://github.com/Trinea/android-open-project/tree/master/English%20Version

## 4.基于ViewGroup，完全定制化

一般用于应用的整体视觉框架，完全定制化，工作量较大，需要考虑的问题较多。  
框架考虑**易用性、扩展性、兼容性**。  
1）**易用性**：保持与android原生控件初始化、改变属性等方式基本一致，降低学习成本。  
例如可以从xml解析，有自己的LayoutParams, 设置焦点、背景等方法。  
2）**扩展性**：框架设计保持弹性，最好可以模块化。  
设计原则，焦点及外框与内容分离，原图与倒影分离，内容为可扩展框架  
3）**兼容性**：对于可扩展框架的onMeasure和onLayout等方法要考虑多种情况。

*待续：Touch事件、Key事件的传递及应用；View Animation总结*

参考文献：

1 《视觉组件库使用说明.docx》

2 Custom Component: <http://developer.android.com/guide/topics/ui/custom-components.html>

3 《Android内核剖析》 柯元旦 View工作原理

