Android 如何绘制视图?

为了更好地理解 ConstraintLayout 的性能，我们先回过头来看看 Android 如何绘制视图。

当用户将某个 Android 视图作为焦点时，Android 框架会指示该视图进行自我绘制。这个绘制过程包括 3 个阶段：

1. 测量

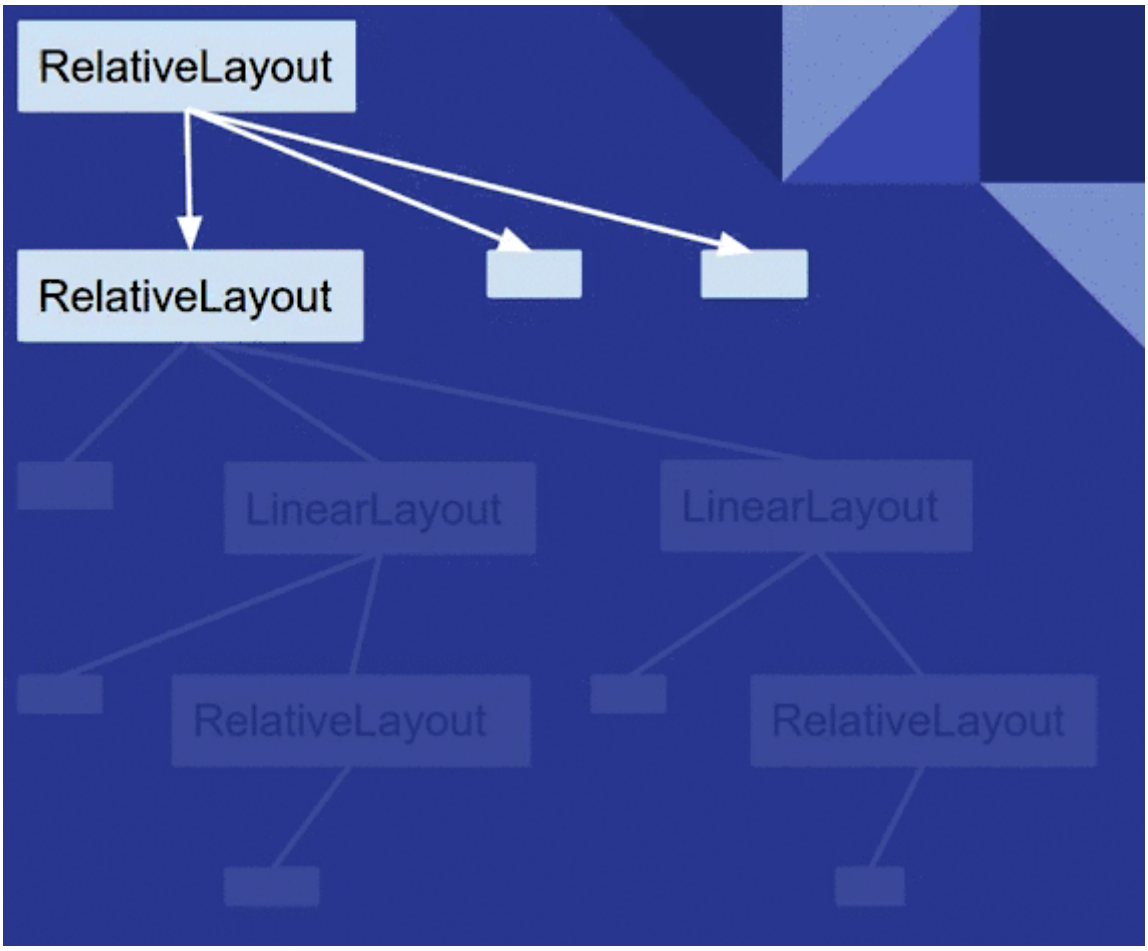
系统自顶向下遍历视图树，以确定每个 ViewGroup 和 View 元素应当有多大。在测量 ViewGroup 的同时也会测量其子对象。

2. 布局

系统执行另一个自顶向下的遍历操作，每个 ViewGroup 都根据测量阶段中所确定的大小来确定其子对象的位置。

3. 绘制

系统再次执行一个自顶向下的遍历操作。对于视图树中的每个对象，系统会为其创建一个 Canvas 对象，以便向 GPU 发送一个绘制命令列表。这些命令包含系统在前面 2 个阶段中确定的 ViewGroup 和 View 对象的大小和位置。

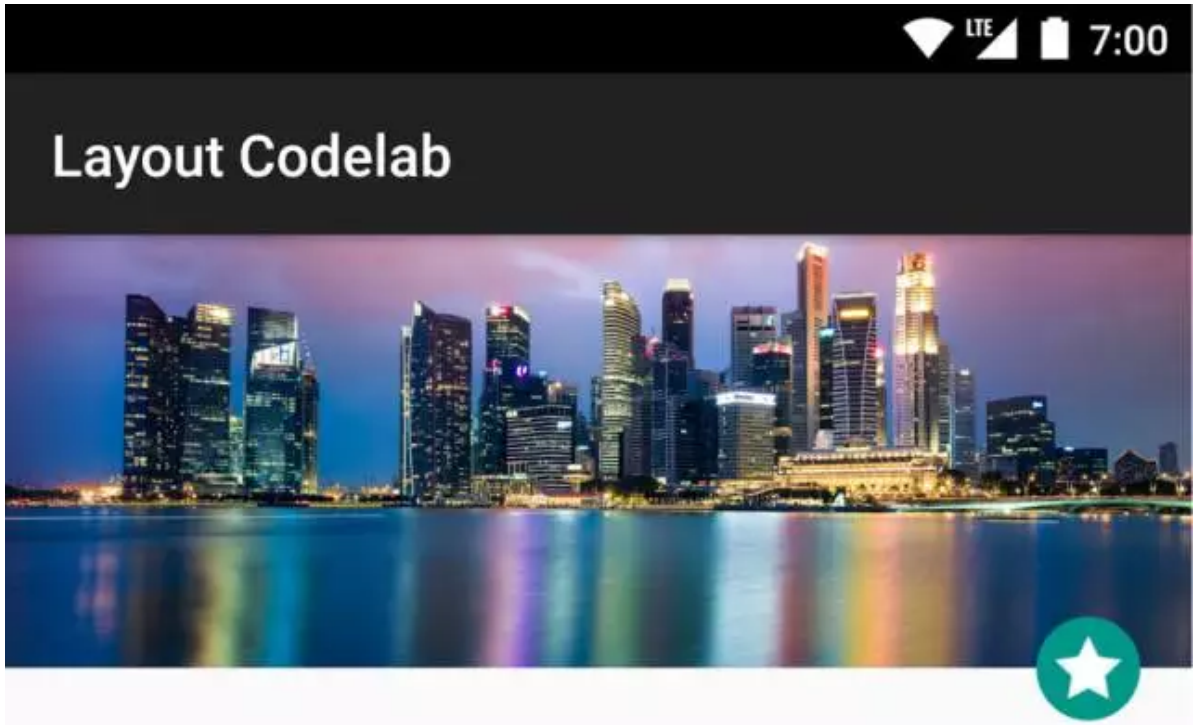


▲ 测量阶段如何遍历视图树的示例

绘制过程中的每个阶段都需要对视图树执行一次自顶向下的遍历操作。因此，视图层次结构中嵌入（或嵌套）的视图越多，设备绘制视图所需的时间和计算功耗也就越多。通过在 Android 应用布局中保持扁平的层次结构，您可以为应用创建响应快速而灵敏的界面。

传统布局层次结构的开销

请牢记上述解释，下面我们来创建一个使用 LinearLayout 和 RelativeLayout 对象的传统布局层次结构。



Singapore

Camera Leica M Typ 240

Settings ƒ/4 16s ISO 200

Singapore officially the Republic of Singapore, and often referred to as the Lion City, the Garden City, and the Red Dot, is a global city in Southeast Asia and the world's only island city-state. It lies one degree (137 km) north of the equator, at the southernmost tip of continental Asia and peninsular Malaysia, with Indonesia's Riau Islands to the south. Singapore's territory consists of the diamond-shaped main island and 62 islets.

UPLOAD DISCARD

▲ 布局示例

假设我们想构建一个像上图那样的布局。如果您使用传统布局来构建，XML 文件会包含类似于下面这样的元素层次结构（在本例中，我们忽略属性）：

```
<RelativeLayout>
  <ImageView />
  <ImageView />
  <RelativeLayout>
    <TextView />
    <LinearLayout>
      <TextView />
      <RelativeLayout>
        <EditText />
      </RelativeLayout>
    </LinearLayout>
    <LinearLayout>
      <TextView />
      <RelativeLayout>
        <EditText />
      </RelativeLayout>
    </LinearLayout>
    <TextView />
  </RelativeLayout>
  <LinearLayout >
    <Button />
    <Button />
  </LinearLayout>
</RelativeLayout>
```

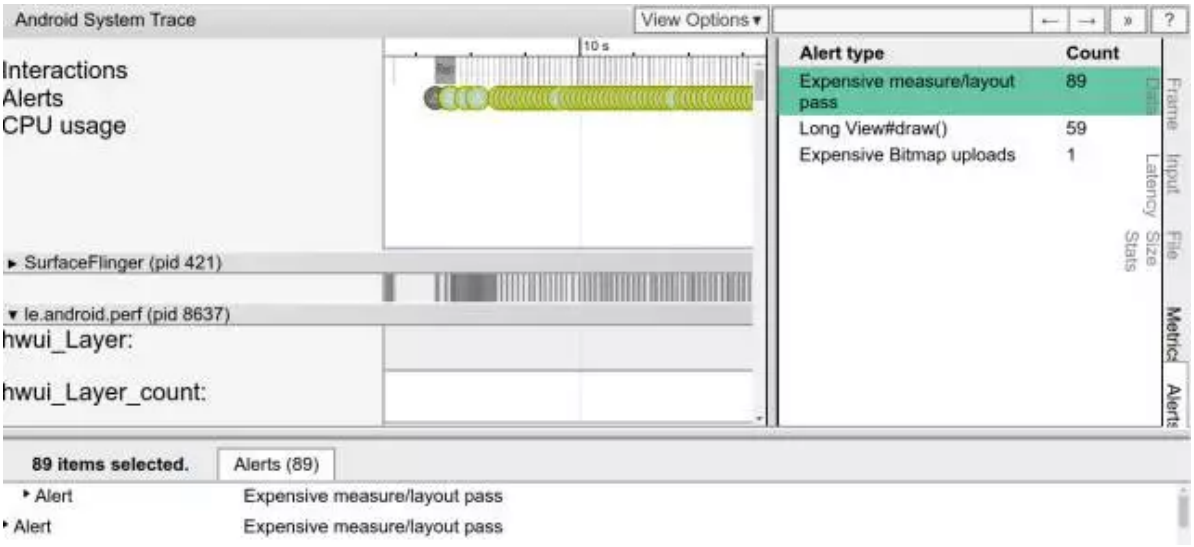
尽管一般来说，这种类型的视图层次结构都有改进的空间，但您几乎必定还需要创建一个包含一些嵌套视图的层次结构。

如前所述，嵌套的层次结构会给性能造成负面影响。我们使用 Android Studio 的 Systrace 工具来看看嵌套视图对界面性能到底有何实际影响。我们通过编程方式针对每个 ViewGroup（ConstraintLayout 和 RelativeLayout）调用了测量和布局阶段并在执行测量和布局调用期间触发了 Systrace。以下命令可生成一个包含 20 秒间隔周期内发生的关键 Event 的概览文件，例如开销巨大的测量/布局阶段：

```
python $ANDROID_HOME/platform-tools/systrace/systrace.py --time=20 -o
~/trace.html gfx view res
```

Systrace 会自动突出显示此布局中的（大量）性能问题，并给出修复这些问题的建议。通过点击“Alerts”标签，您会发现，绘制此视图层次结构需要反复执行 80 次的测量和布局阶段，开销极为庞大！

触发开销如此庞大的测量和布局阶段当然很不理想，如此庞大的绘制 Activity 会导致用户能够觉察到丢帧的现象。我们可以得出这样的结论：这种嵌套式层次结构和 RelativeLayout（会对其每个子对象重复测量两次）的特性导致性能低下。



▲ 观察 Systrace 针对使用 RelativeLayout 的布局版本发出的提醒

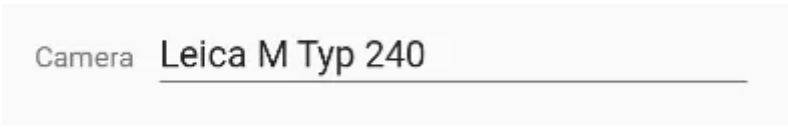
ConstraintLayout 对象的优势

如果您使用 ConstraintLayout 来构建相同的布局，XML 文件会包含类似于下面这样的元素层次结构（再次忽略属性）：

```
<android.support.constraint.ConstraintLayout>
  <ImageView />
  <ImageView />
  <TextView />
  <EditText />
  <TextView />
  <TextView />
  <EditText />
  <Button />
  <Button />
  <TextView />
</android.support.constraint.ConstraintLayout>
```

如本例所示，现在，该布局拥有一个完全扁平的层次结构。这是因为 ConstraintLayout 允许您构建复杂的布局，而不必嵌套 View 和 ViewGroup 元素。

举个例子，我们来看一下布局中间的 TextView 和 EditText：



使用 RelativeLayout 时，您需要创建一个新的 ViewGroup 来垂直对齐 EditText 和 TextView：

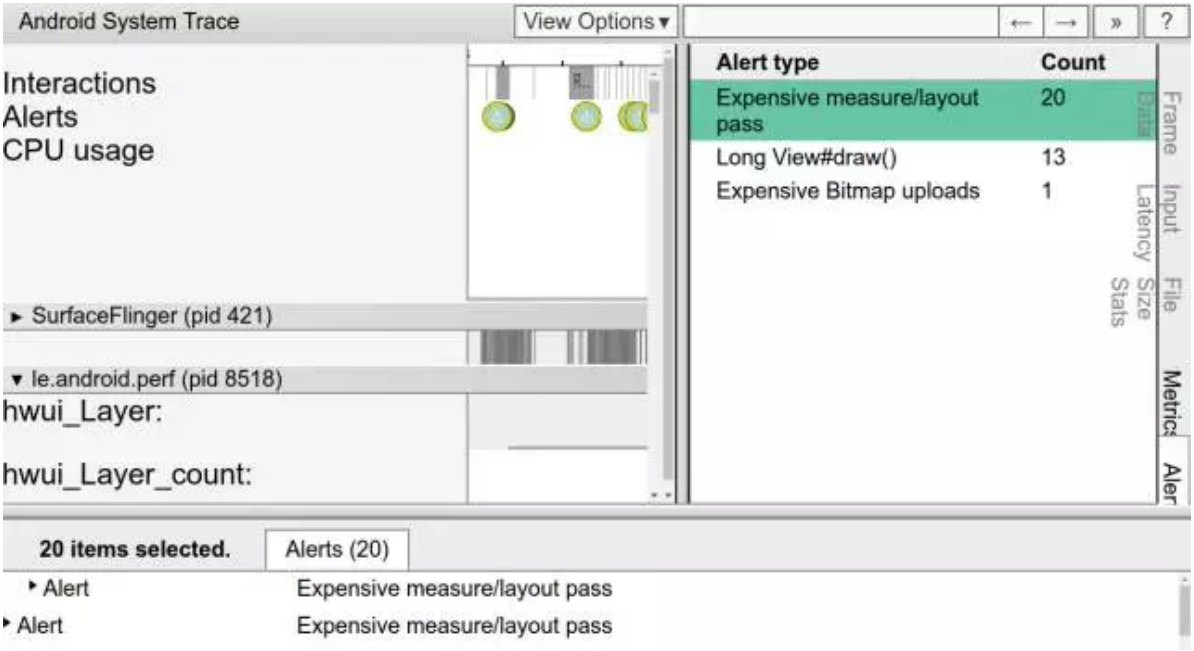
```
<LinearLayout
    android:id="@+id/camera_area"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_below="@id/title" >

    <TextView
        android:text="@string/camera"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_gravity="center_vertical"
        android:id="@+id/cameraLabel"
        android:labelFor="@+id/cameraType"
        android:layout_marginStart="16dp" />

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <EditText
            android:id="@+id/cameraType"
            android:ems="10"
            android:inputType="textPersonName"
            android:text="@string/camera_value"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"
            android:layout_marginTop="8dp"
            android:layout_marginStart="8dp"
            android:layout_marginEnd="8dp" />
    </RelativeLayout>
</LinearLayout>
```

通过改用 ConstraintLayout，您只需添加一个从 TextView 基线到 EditText 基线之间的约束，即可实现同样的效果，而不必创建另一个 ViewGroup：

▲ EditText 和 TextView 之间的约束

在针对我们使用 ConstraintLayout 的布局版本运行 Systrace 工具时，您会发现，同样 20 秒间隔周期内执行的测量/布局次数大大减少，开销也随之大大减少。这种性能的改进很有意义，现在，我们保持了扁平的视图层次结构！



▲ 观察 Systrace 针对使用 ConstraintLayout 的布局版本发出的提醒

同样值得一提的是，我们构建 ConstraintLayout 版本的布局时仅仅使用了布局编辑器，而不是手工编辑 XML。而要使用 RelativeLayout 来实现同样的视觉效果，我们很可能必须手工编辑 XML。

测量性能差异

我们使用 Android 7.0（API 级别 24）中引入的 OnFrameMetricsAvailableListener 分析了 ConstraintLayout 和 RelativeLayout 这两种类型的布局所执行的每次测量和布局操作所花费的时间。通过该类，您可以收集有关应用界面渲染的逐帧时间信息。

通过调用以下代码，您可以开始记录每个帧的界面操作：

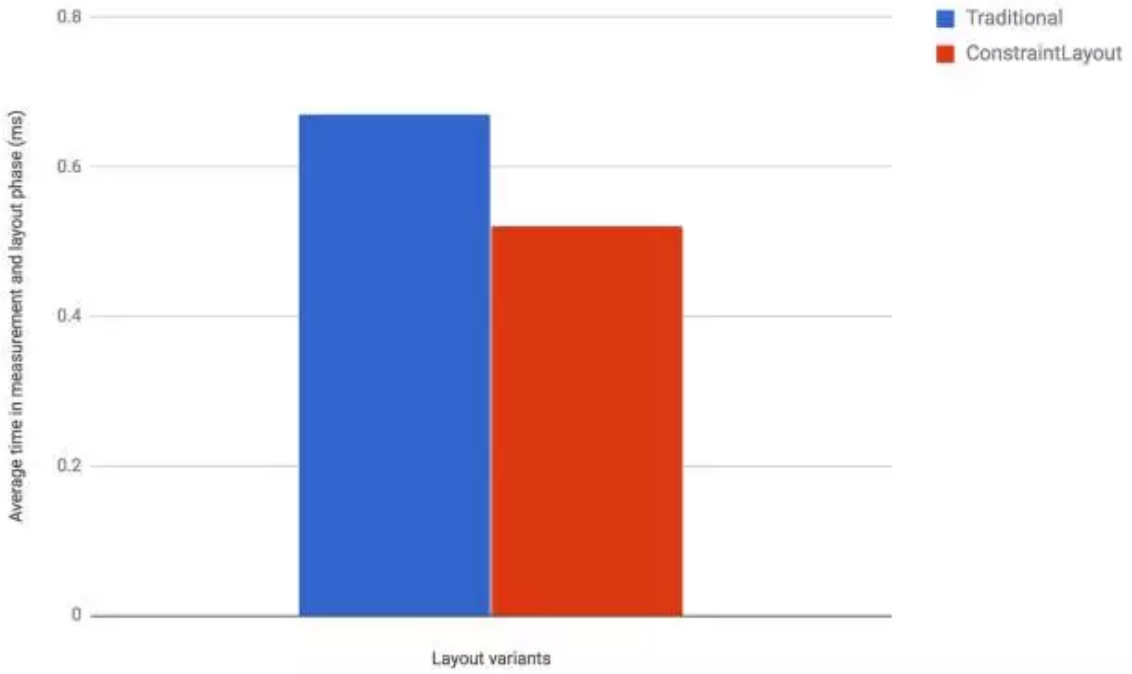
```
window.addOnFrameMetricsAvailableListener(
        frameMetricsAvailableListener, frameMetricsHandler);
```

在能够获取时间信息之后，该应用触发 frameMetricsAvailableListener() 回调。我们对测量/布局的性能感兴趣，因此，我们在检索实际帧的持续时间时调用了 FrameMetrics.LAYOUT_MEASURE_DURATION。

```
Window.OnFrameMetricsAvailableListener {
        _, frameMetrics, _ ->
        val frameMetricsCopy = FrameMetrics(frameMetrics);
        // Layout measure duration in nanoseconds
        val layoutMeasureDurationNs =

frameMetricsCopy.getMetric(FrameMetrics.LAYOUT_MEASURE_DURATION);
```

测量结果：ConstraintLayout 速度更快

我们的性能比较结果表明：ConstraintLayout 在测量/布局阶段的性能比 RelativeLayout大约高 40%：



▲ 测量/布局（单位：毫秒，100 帧的平均值）

这些结果表明：ConstraintLayout 很可能比传统布局的性能更出色。不仅如此，ConstraintLayout 还具备其他一些功能，能够帮助您构建复杂的高性能布局。

我们建议您在设计应用布局时使用 ConstraintLayout。在过去，几乎所有情形下，您都需要一个深度嵌套的布局，因此，ConstraintLayout 应当成为您优化性能和易用性的不二之选。