



作者 尹star (/users/bd3befbe51d0) 2015.12.16 21:12\*

写了51093字，被881人关注，获得了1374个喜欢  
(/users/bd3befbe51d0)

+ | 添加关注 (/sign\_in)

# Android性能优化之如何避免Overdraw

字数3001 阅读4780 评论12 喜欢76

## 什么是Overdraw？

Overdraw就是过度绘制，是指在一帧的时间内（16.67ms）像素被绘制了多次，理论上一个像素每次只绘制一次是最优的，但是由于重叠的布局导致一些像素会被多次绘制，而每次绘制都会对应到CPU的一组绘图命令和GPU的一些操作，当这个操作耗时超过16.67ms时，就会出现掉帧现象，也就是我们所说的卡顿，所以对重叠不可见元素的重复绘制会产生额外的开销，需要尽量减少Overdraw的发生。

Android提供了测量Overdraw的选项，在开发者选项—调试GPU过度绘制（Show GPU Overdraw），打开选项就可以看到当前页面Overdraw的状态，就可以观察屏幕的绘制状态。该工具会使用三种不同的颜色绘制屏幕，来指示overdraw发生在哪以及程度如何，其中：

没有颜色：意味着没有overdraw。像素只画了一次。

蓝色：意味着overdraw 1倍。像素绘制了两次。大片的蓝色还是可以接受的（若整个窗口是蓝色的，可以摆脱一层）。

绿色：意味着overdraw 2倍。像素绘制了三次。中等大小的绿色区域是可以接受的但你应该尝试优化、减少它们。

浅红：意味着overdraw 3倍。像素绘制了四次，小范围可以接受。

暗红：意味着overdraw 4倍。像素绘制了五次或者更多。这是错误的，要修复它们。



552dd397d5e53.jpg



那么我们怎么来消灭overdraw呢？总的原则就是：尽量避免重叠不可见元素的绘制，基于这个原则，我们大概可以想出以下几招：

## 第一招：合理选择控件容器

既然overdraw是因为重复绘制了同一片区域的像素点，那我们首先想到的是解决布局问题。Android提供的Layout控件主要包括LinearLayout、TableLayout、FrameLayout、RelativeLayout。俗话说条条大路通罗马，同一个界面我们可以使用不同的容器控件来表达，但是各个容器控件描述界面的复杂度是不一样的。一般来说LinearLayout最易，RelativeLayout较复杂。但是尺有所短，寸有所长，LinearLayout只能用来描述一个方向上连续排列的控件，而RelativeLayout几乎可以用于描述任意复杂度的界面。但是我要说但是了，表达能力越强的容器控件，性能往往略低一些，因为系统需要将更多的时间花在计算子控件的位置上。综上所述：**LinearLayout易用，效率高，表达能力有限。RelativeLayout复杂，表达能力强，效率稍逊。**

那么对于同一界面而言，作为开发者考虑是使用尽量少的、表达能力强的RelativeLayout作为容器，还是选择多个、表达能力稍弱的LinearLayout来展示。从减少overdraw的角度来看，LinearLayout会增加控件数的层级，自然是RelativeLayout更优，但是当某一界面在使用LinearLayout并不会比RelativeLayout带来更多的控件数和控件层级时，LinearLayout则是首选。所以在表达界面的时候，作为一个有前瞻性的开发者要根据实际情况来选择合适容器控件，在保证性能的同时，尽量避免overdraw。

## 第二招：去掉window的默认背景

当我们使用了Android自带的一些主题时，window会被默认添加一个纯色的背景，这个背景是被DecorView持有的。当我们的自定义布局时又添加了一张背景图或者设置背景色，那么DecorView的background此时对我们来说是无用的，但是它会产生一次Overdraw，带来绘制性能损耗。

去掉window的背景可以在onCreate()中setContentView()之后调用

```
getWindow().setBackgroundDrawable(null);
```

或者在theme中添加

```
android:windowbackground="null";
```

## 第三招：去掉其他不必要的背景

有时候为了方便会先给Layout设置一个整体的背景，再给子View设置背景，这里也会造成重叠，如果子View宽度match\_parent，可以看到完全覆盖了Layout的一部分，这里就可以通过分别设置背景来减少重绘。再比如如果采用的是selector的背景，将normal状态的color设置为“@android:color/transparent”，也同样可以解决问题。这里只简单举两个例子，我们在开发过程中的一些习惯性思维定式会带来不经意的Overdraw，所以开发过程中我们为某个View或者ViewGroup设置背景的时候，先思考下是否真的有必要，或者思考下这个背景能不能分段设置在子View上，而不是图方便直接设置在根View上。

## 第四招：ClipRect & QuickReject

为了解决Overdraw的问题，Android系统会通过避免绘制那些完全不可见的组件来尽量减少消耗。但是不幸的是，对于那些过于复杂的自定义的View(通常重写了onDraw方法)，Android系统无法检测在onDraw里面具体会执行什么操作，系统无法监控并自动优化，也就无法避免Overdraw了。但是我们可以通过

canvas.clipRect()来帮助系统识别那些可见的区域。这个方法可以指定一块矩形区域，只有在这个区域内才会被绘制，其他的区域会被忽视。这个API可以帮助那些有多组重叠组件的自定义View来控制显示的区域。同时clipRect方法还可以帮助节约CPU与GPU资源，在clipRect区域之外的绘制指令都不会被执行，那些部分内容在矩形区域内的组件，仍然会得到绘制。除了clipRect方法之外，我们还可以使用canvas.quickreject()来判断是否没和某个矩形相交，从而跳过那些非矩形区域内的绘制操作。

## 第五招：ViewStub

ViewStub是个什么东西？一句话总结：**高效占位符**。

我们经常会遇到这样的情况，运行时动态根据条件来决定显示哪个View或布局。常用的做法是把View都写在上面，先把它们的可见性都设为View.GONE，然后在代码中动态的更改它的可见性。这样的做法的优点是逻辑简单而且控制起来比较灵活。但是它的缺点就是，耗费资源。虽然把View的初始可见View.GONE但是在Inflate布局的时候View仍然会被Inflate，也就是说仍然会创建对象，会被实例化，会被设置属性。也就是说，会耗费内存等资源。

推荐的做法是使用android.view.ViewStub，ViewStub是一个轻量级的View，它一个看不见的，不占布局位置，占用资源非常小的控件。可以为ViewStub指定一个布局，在Inflate布局的时候，只有ViewStub会被初始化，然后当ViewStub被设置为可见的时候，或是调用了ViewStub.inflate()的时候，ViewStub所向的布局就会被Inflate和实例化，然后ViewStub的布局属性都会传给它所指向的布局。这样，就可以使用ViewStub来方便的在运行时，要还是不要显示某个布局。

```
<ViewStub  
    android:id="@+id/stub_view"  
    android:inflatedId="@+id/panel_stub"  
    android:layout="@layout/progress_overlay"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_gravity="bottom" />
```

当你想加载布局时，可以使用下面其中一种方法：

```
((ViewStub) findViewById(R.id.stub_view)).setVisibility(View.VISIBLE);  
View importPanel = ((ViewStub) findViewById(R.id.stub_view)).inflate();
```

## 第六招：Merge

Merge标签有什么用呢？简单粗暴点回答：**干掉一个view层级**。

Merge的作用很明显，但是也有一些使用条件的限制。有两种情况下我们可以使用Merge标签来做容器控件。第一种子视图不需要指定任何针对父视图的布局属性，就是说父容器仅仅是个容器，子视图只需要直接添加到父视图上用于显示就行。另外一种是假如需要在LinearLayout里面嵌入一个布局（或者视图），而恰恰这个布局（或者视图）的根节点也是LinearLayout，这样就多了一层没有用的嵌套，无疑这样只会拖慢程序速度。而这个时候如果我们使用merge根标签就可以避免那样的问题。另外Merge只能作为XML布局的根标签使用，当Inflate以<merge />开头的布局文件时，必须指定一个父ViewGroup，并且必须设定attachToRoot为true。

举个简单的例子吧：

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="merge标签使用" />
</RelativeLayout>
```

把上面这个XML加载到页面中，布局层级是RelativeLayout-TextView。但是采用下面的方式，把RelativeLayout替换成merge，RelativeLayout这一层级就被干掉了。

```
<merge
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="merge标签使用" />
</merge>
```

## 第七招：善用draw9patch

给ImageView加一个边框，你肯定遇到过这种需求，通常在ImageView后面设置一张背景图，露出边框便完美解决问题，此时这个ImageView，设置了两层drawable，底下一层仅仅是为了作为图片的边框而已。但是两层drawable的重叠区域去绘制了两次，导致overdraw。

优化方案：将背景drawable制作成draw9patch，并且将和前景重叠的部分设置为透明。由于Android的2D渲染器会优化draw9patch中的透明区域，从而优化了这次overdraw。但是背景图片必须制作成draw9patch才行，因为Android 2D渲染器只对draw9patch有这个优化，否则，一张普通的Png，就算你把中间的部分设置成透明，也不会减少这次overdraw。

## 第八招：慎用Alpha

假如对一个View做Alpha转化，需要先将View绘制出来，然后做Alpha转化，最后将转换后的效果绘制在界面上。通俗点说，做Alpha转化就需要对当前View绘制两遍，可想而知，绘制效率会大打折扣，耗时会翻倍，所以Alpha还是慎用。

如果一定做Alpha转化的话，可以采用缓存的方式。

```
view.setLayerType(LAYER_TYPE_HARDWARE);
doSmoeThing();
view.setLayerType(LAYER_TYPE_NONE);
```

通过setLayerType方式可以将当前界面缓存在GPU中，这样不需要每次绘制原始界面，但是GPU内存是相当宝贵的，所以用完要马上释放掉。

## 第九招：避免“OverDesign”

overdraw会给APP带来不好的体验，overdraw产生的原因无外乎：复杂的Layout层级，重叠的View，重叠的背景这几种。开发人员无节制的View堆砌，究其根本无非是产品无节制的需求设计。有道是“由俭入奢易，由奢入俭难”，很多APP披着过度设计的华丽外衣，却忘了简单易用才是王道的本质，纷繁复杂的设计并不会给用户带来好的体验，反而会让用户有压迫感，产品本身也有可能因此变得卡顿。当然，一切抛开业务谈优化都是空中楼阁，这就需要产品设计也要有一个权衡，在复杂的业务逻辑与简单易用的界面展现中做一个平衡，而不是一味的OverDesign。

⊕ 推荐拓展阅读 (/sign\_in)

© 著作权归作者所有

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作！

¥ 打赏支持

♥ 喜欢 | 76

微博 分享到微博 微信 分享到微信  
更多分享 ▾

12条评论 ( 按时间正序 · 按时间倒序 · 按喜欢排序 )

添加新评论 (/sign\_in)

eclipse\_xu (/users/dfc0ed52c22b)  
(/users/dfc0ed52c22b) 2015.12.17 11:39 (/p/145fc61011cd/comments/1050099#comment-1050099)

RelativeLayout复杂，表达能力强，效率低。作者是如何得出这个结论的

喜欢(0)

回复

尹star (/users/bd3befbe51d0): @eclipse\_xu (/users/dfc0ed52c22b) 书上看到的，有什么问题么。  
2015.12.17 11:33 (/p/145fc61011cd/comments/1050124#comment-1050124)

回复

eclipse\_xu (/users/dfc0ed52c22b): @尹star (/users/bd3befbe51d0) 在绘制效率和计算效率上，rl是高于ll的，这种优势，布局深度越深，优势越明显  
2015.12.17 12:36 (/p/145fc61011cd/comments/1050431#comment-1050431)

回复

jave1987 (/users/84b10d474c93): @eclipse\_xu (/users/dfc0ed52c22b) 应该是相比较而言多了一次绘制  
2015.12.17 12:39 (/p/145fc61011cd/comments/1050456#comment-1050456)

回复

还有 4 条评论，展开查看

添加新回复

自导自演的机器人 (/users/cfcd1814672c)  
(/users/cfcd1814672c) 2015.12.17 21:45 (/p/145fc61011cd/comments/1053328#comment-1053328)

回复

喜欢(0)

回复

 android\_ccy (/users/5d7e450ce27b)  
(/users/5d7e450ce27b) 2016-07-18 17:03 (/p/145fc61011cd/comments/1057348#comment-1057348)

路过... 

 喜欢(0)

回复

 往山 (/users/8e60c4c603ac)  
(/users/8e60c4c603ac) 2016-07-18 15:27 (/p/145fc61011cd/comments/3170199#comment-3170199)

请问你的第7招draw9patch具体是怎么使用呢？我写了个小例子：两个ImageView，分别设置src和background，其中一张的background按你所说设置成了重叠地方透明。但是效果不明显。

 喜欢(0)

回复

 Arison (/users/7fcf7246c714)  
(/users/7fcf7246c714) 2016-07-26 07:49 (/p/145fc61011cd/comments/3288088#comment-3288088)

说的不错

 喜欢(0)

回复

[登录后发表评论 \(/sign\\_in\)](#)

被以下专题收入，发现更多相似内容：

### **Android知识 (/collection/3fde3b545a35)**

分享Android开发的知识，教程，解析，前沿信息，都可以，欢迎大家投稿~ 内容可搞笑，可逗比，另外欢迎  
(/collection/3fde3b545a35)  | [添加关注 \(/sign\\_in\)](#)

2969篇文章 (/collection/3fde3b545a35) · 18140人关注

### **Android开发经验谈 (/collection/5139d555c94d)**

Android老鸟给新人的建议、资源。更优质的原创内容，欢迎关注技术公众号，微信搜索：“Open软件开发”  
(/collection/5139d555c94d)  | [添加关注 \(/sign\\_in\)](#)

2624篇文章 (/collection/5139d555c94d) · 10236人关注

### **Android技术知识 (/collection/58b4c20abf2f)**

Android深入理解、基础详解及各种Library使用介绍。认真做技术，好好享受人生。。  
(/collection/58b4c20abf2f)  | [添加关注 \(/sign\\_in\)](#)

2299篇文章 (/collection/58b4c20abf2f) · 2804人关注