#### 阿里巴巴 给你一个Demo 你如何快速定位ANR

本专栏专注分享大型Bat面试知识，后续会持续更新，喜欢的话麻烦点击一个star

##### 一、前期基础知识储备

（1）**ANR错误定义**：在Android上，如果你的应用程序有一段时间响应不够灵敏，系统会向用户显示一个对话框，这个对话框称作“应用程序无响应”（ANR：Application Not Responding）对话框。用户可以选择“等待”而让程序继续运行，也可以选择“强制关闭”。因此，在程序里对响应性能的设计很重要，这样，系统不会显示ANR给用户。

默认情况下，在Android中Activity的最长执行时间是5秒（主要类型），BroadcastReceiver的最长执行时间的则是10秒，ServiceTimeout的最长执行时间是20秒（少数类型）。超出就会提示应用程序无响应（ANR错误）。

​ 

（2）**ANR错误出现原因**：只有当应用程序的UI线程响应超时才会引起ANR 超时产生的原因包括：①当前事件没有机会处理，例如UI线程正在响应另外的事件，当前事件被某个事件给阻塞掉了；②当前事件正在处理 但是由于耗时太长没有能及时的完成。其他原因：③在BroadcastReceiver里做耗时的操作或计算；④CPU使用过高；⑤发生了死锁；⑥耗时操作的动画需要大量的计算工作，可能导致CPU负载过重。

##### 二、ANR定位方式及优化

（1）ANR错误定位——如果开发机器上出现ANR问题时，系统会生成一个traces.txt的文件放在/data/anr下，最新的ANR信息在最开始部分。通过adb命令将其导出到本地，输入以下字符：

$adb pull data/anr/traces.txt .

（2）**供选的优化ANR问题的方式：**

1）为了执行一个长时间的耗时操作而创建一个工作线程最方便高效的方式是使用AsyncTask，只需要继承AsyncTask并实现doInBackground()方法来执行任务即可。为了把任务执行的进度呈现给用户，你可以执行publishProgress()方法，这个方法会触发onProgressUpdate()的回调方法。在onProgressUpdate()的回调方法中(它执行在UI线程)，你可以执行通知用户进度的操作，例如：

private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {  
 // Do the long-running work in here  
 protected Long doInBackground(URL... urls) {  
 int count = urls.length;  
 long totalSize = 0;  
 for (int i = 0; i < count; i++) {  
 totalSize += Downloader.downloadFile(urls[i]);  
 publishProgress((int) ((i / (float) count) \* 100));  
 // Escape early if cancel() is called  
 if (isCancelled()) break;  
 }  
 return totalSize;  
 }  
   
 // This is called each time you call publishProgress()  
protected void onProgressUpdate(Integer... progress) {  
 setProgressPercent(progress[0]);  
}  
  
// This is called when doInBackground() is finished  
protected void onPostExecute(Long result) {  
 showNotification("Downloaded " + result + " bytes");  
}

2）如果你实现了Thread或者HandlerThread，请确保你的UI线程不会因为等待工作线程的某个任务而去执行Thread.wait()或者Thread.sleep()。UI线程不应该去等待工作线程完成某个任务，你的UI线程应该提供一个Handler给其他工作线程，这样工作线程能够通过这个Handler在任务结束的时候通知UI线程。例如：

**继承Thread类**

new Thread(new Runnable() {  
 @Override  
 public void run() {  
 /\*\*  
 耗时操作  
 \*/  
 handler.post(new Runnable() {  
 @Override  
 public void run() {  
 /\*\*  
 更新UI  
 \*/  
 }  
 });  
 }  
 }).start();

**实现Runnable接口**

class PrimeRun implements Runnable {  
 long minPrime;  
 PrimeRun(long minPrime) {  
 this.minPrime = minPrime;  
 }  
  
 public void run() {  
 // compute primes larger than minPrime  
 . . .  
 }  
}  
  
PrimeRun p = new PrimeRun(143);  
new Thread(p).start();

**使用HandlerThread**

// 启动一个名为new\_thread的子线程  
HandlerThread thread = new HandlerThread("new\_thread");  
thread.start();  
  
// 取new\_thread赋值给ServiceHandler  
private ServiceHandler mServiceHandler;  
mServiceLooper = thread.getLooper();  
mServiceHandler = new ServiceHandler(mServiceLooper);  
  
private final class ServiceHandler extends Handler {  
 public ServiceHandler(Looper looper) {  
 super(looper);  
 }  
   
 @Override  
 public void handleMessage(Message msg) {  
 //默认Handler的handleMessage方法是运行在主线程中的，如果传入一个工作线程的Looper，则改变HandleMessage方法执行的所在线程  
 }  
}

3）开发在日常的开发过程中使用Thread或者HandlerThread，可以尝试调用Process.setThreadPriority(Process.THREAD\_PRIORITY\_BACKGROUND)设置较低的优先级，否则仍然会降低程序响应，因为默认Thread的优先级和主线程相同。

4）Activity的onCreate和onResume回调中尽量避免耗时的代码，应该尽可能的做比较少的事情，其实，任何执行在UI线程中的方法都应该尽可能简短快速。类似网络或者DB操作等可能长时间执行的操作，或者是类似调整bitmap大小等需要长时间计算的操作，都应该执行在工作线程中。

5）BroadcastReceiver中onReceive代码也要尽量减少耗时。如果必须在onReceive方法中执行耗时操作，建议使用IntentService进行处理，IntentService集开启线程和自动关闭服务两种功能于一身，本身非常灵活。

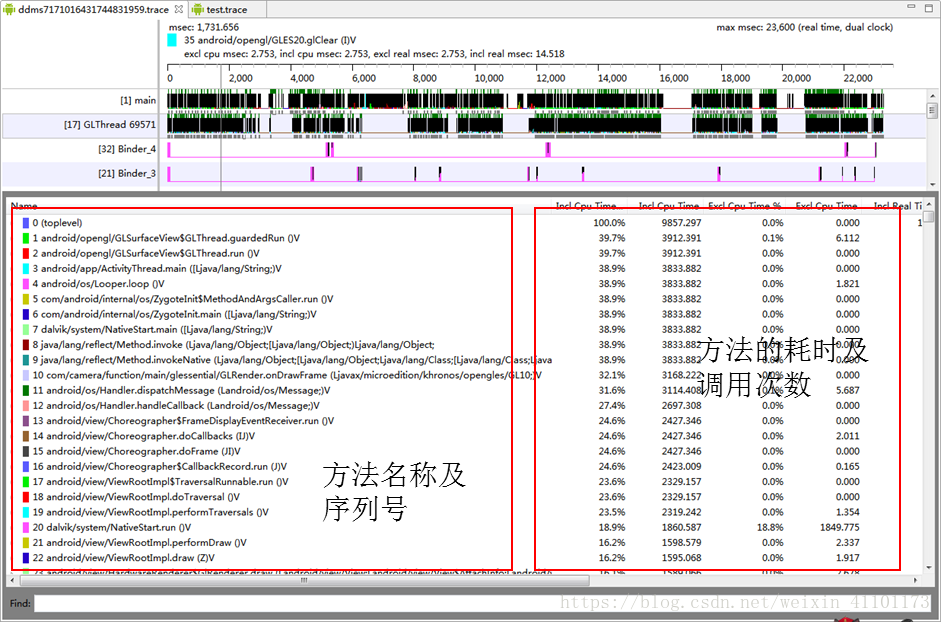
@Override  
public void onReceive(Context context, Intent intent) {  
 // This is a long-running operation  
 BubbleSort.sort(data);  
}  
//上面的代码在onReceive方法中执行了耗时操作

@Override  
public void onReceive(Context context, Intent intent) {  
 // The task now runs on a worker thread.  
 Intent intentService = new Intent(context, MyIntentService.class);  
 context.startService(intentService);  
}  
  
public class MyIntentService extends IntentService {  
 @Override  
 protected void onHandleIntent(@Nullable Intent intent) {  
 BubbleSort.sort(data);  
 }  
}  
//将onReceive的耗时操作放入到IntentService中执行，执行完之后自动关闭工作线程

6）增加界面响应性（交互层面），这是一个成熟应用必备的标志—通常来说，100ms - 200ms是用户能够察觉到卡顿的上限。如果你的程序在启动阶段有一个耗时的初始化操作，可以考虑显示一个闪屏，要么尽快的显示主界面，然后马上显示一个加载的对话框，异步加载数据。无论哪种情况，你都应该显示一个进度信息，以免用户感觉程序有卡顿的情况。

##### 三、辅助处理ANR问题的工具

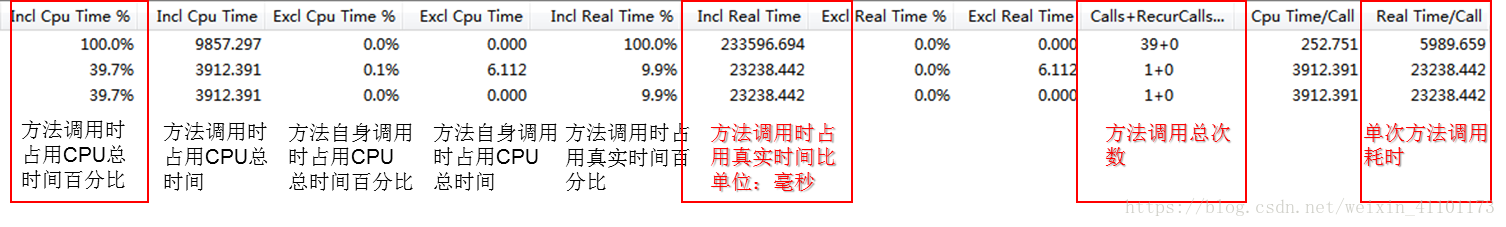
（1）Traceview - 系统性能分析工具，用于定位应用代码中的耗时操作



img

①选好应用的进程，执行一段应用操作，从图中的上半部分，可以看到各个线程的各个方法的执行时间；

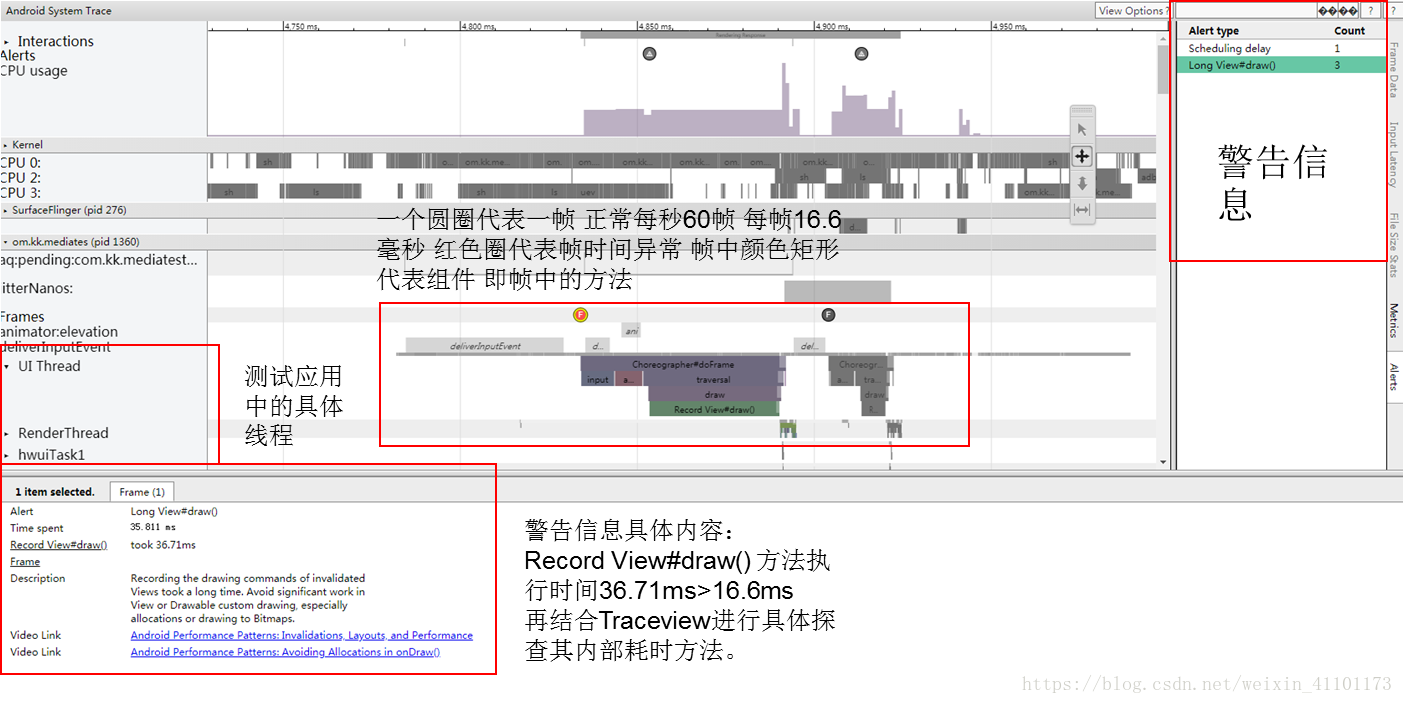
②从图中的下半部分，可以该段操作中具体调用的方法和每个方法的执行时间、执行次数。占CPU的百分比；



img

该图是具体的方法执行时间分布图，我们重点关注其中的“Incl Real Time”这一时间指标，其为方法的实际调用时间，单位毫秒，查看时点击Incl Real Time进行排列，方法会根据时间长短进行排列，其中超过500ms的方法我们都该重点关注。

（2）Systrace - Android4.1新增的应用性能数据采样和分析工具（与google引擎联合开发 使用时借助chorme浏览器）



img

连接手机，进行一段操作，系统会生成一份Html文件，在谷歌浏览器中打开，如图：

①Sytrace会显示在这段操作期间所有的进程信息，在其中找到自己的进程，可以看到在测试进程中，我们定位UI Thread，可以看到里面的系统方法，这是UI渲染时的调用方法，上面有一个个的圈，绿色圈代表帧渲染时间是16.6ms（Android系统渲染UI界面时间为1秒60帧，每帧即16.6ms），超过该值的帧用红色圈标注；

##### ②点击红色圈的标注帧，可以看到Sytrace给出的Alert，具体查看可发现，给出了该帧具体的渲染时间为36.71ms，超过16.6ms，UI渲染时会发生掉帧的情况，即卡顿，再往下看，可以看到系统给出的建议和超时的主要方法。拿到这个方法再结合Traceview工具，进行具体分析，找到自己的代码，进行优化，减少耗时。