

sunzn'Blog

博客园 首页 新随笔 联系 订阅 管理

随笔 - 104 文章 - 0 评论 - 144



Android 编程下的 TraceView 简介及其案例实战

TraceView 是 Android 平台配备一个很好的性能分析的工具。它可以通过图形化的方式让我们了解我们要跟踪的程序的性能，并且能具体到 method。详细内容参考：[Profiling with Traceview and dmtracedump](#)

TraceView 简介

TraceView 是 Android 平台特有的数据采集和分析工具，它主要用于分析 Android 中应用程序的 hotspot。TraceView 本身只是一个数据分析工具，而数据的采集则需要使用 Android SDK 中的 Debug 类或者利用 DDMS 工具。二者的用法如下：

- 开发者在一些关键代码段开始前调用 Android SDK 中 Debug 类的 startMethodTracing 函数，并在关键代码段结束前调用 stopMethodTracing 函数。这两个函数运行过程中将采集运行时间内该应用所有线程（注意，只能是 Java 线程）的函数执行情况，并将采集数据保存到 /mnt/sdcard/ 下的一个文件中。开发者然后需要利用 SDK 中的 TraceView 工具来分析这些数据。
- 借助 Android SDK 中的 DDMS 工具。DDMS 可采集系统中某个正在运行的进程的函数调用信息。对开发者而言，此方法适用于没有目标应用源代码的情况。

DDMS 中 TraceView 使用示意图如下，调试人员可以通过选择 Devices 中的应用后点击  按钮 Start Method Profiling（开启方法分析）和点击  Stop Method Profiling（停止方法分析）

2017年9月						
<	日	一	二	三	四	五
	27	28	29	30	31	1
	3	4	5	6	7	8
	10	11	12	13	14	15
	17	18	19	20	21	22
	24	25	26	27	28	29
	1	2	3	4	5	6

搜索

常用链接

[我的随笔](#)

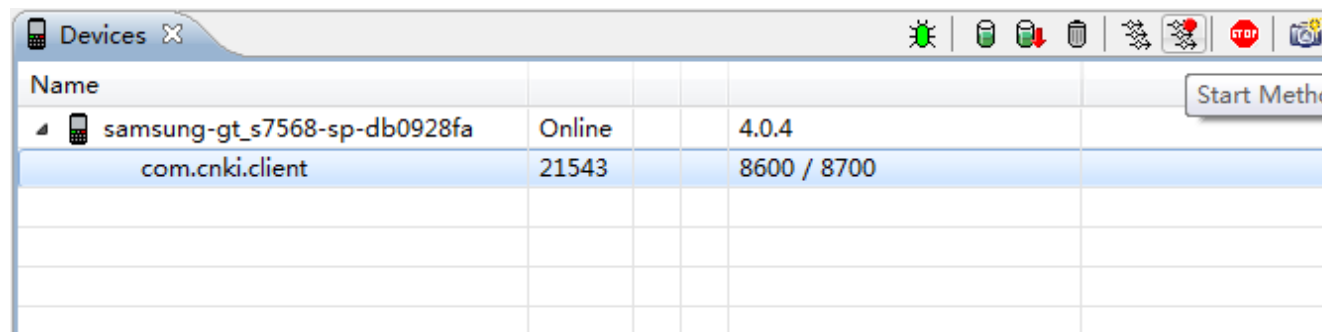
[我的评论](#)

[我的参与](#)

[最新评论](#)

[我的标签](#)

我的标签



开启方法分析后对应用的目标页面进行测试操作，测试完毕后停止方法分析，界面会跳转到 DDMS 的 trace 分析界面，如下图所示：

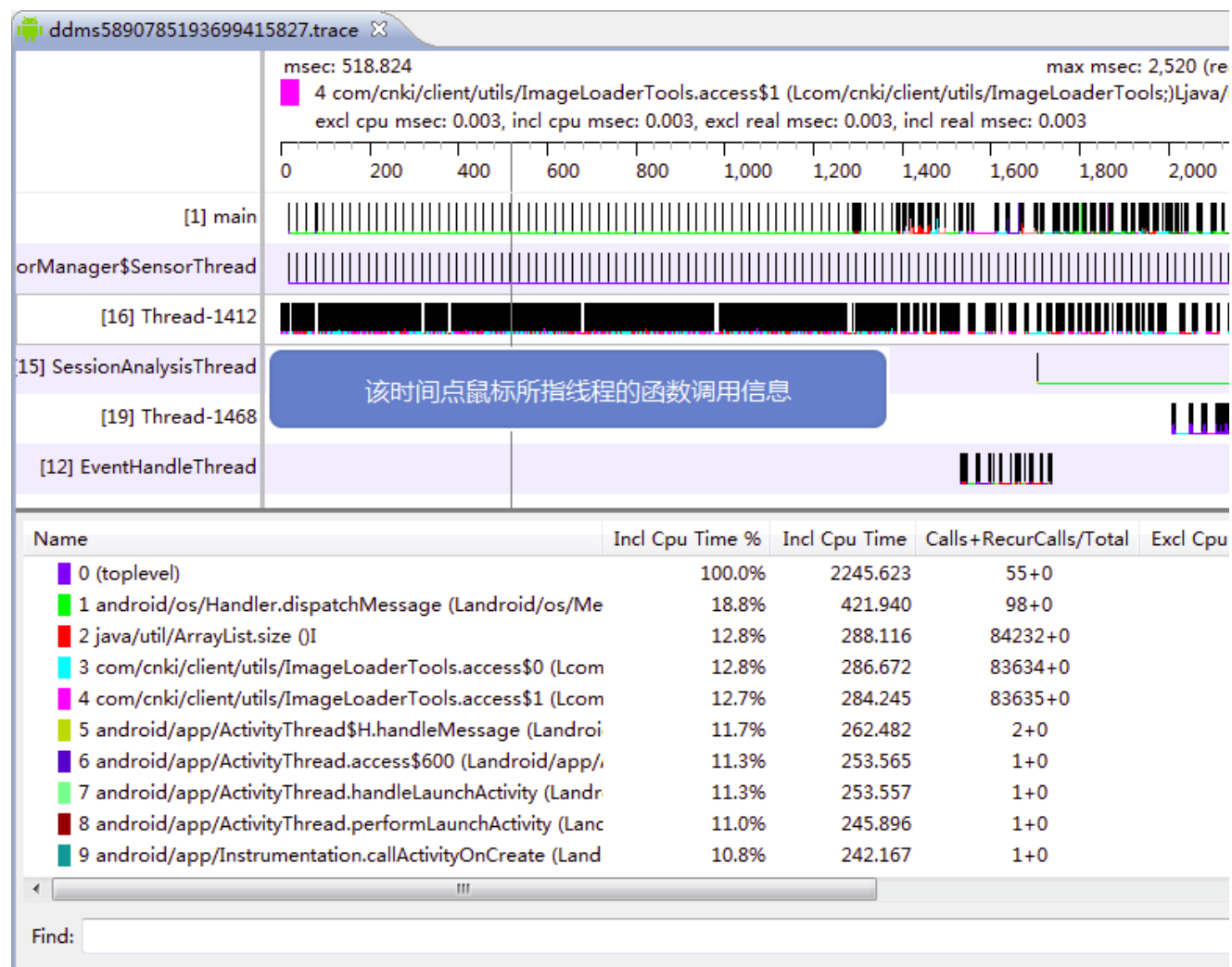
多线程 (6) 工具类 (6) 动画 (5)
ListView (4) 技巧 (4) 内存 (3)
生命周期 (3) 反射 (3) 效果 (3)
优化 (3) 更多

随笔分类

安卓开发(56)
案例源码(9)
编程基础(19)
产品设计(1)
错误收集(5)
官方文档(4)
应用技巧(15)
有趣事物(1)
资源收集(2)

随笔档案

2016年10月 (1)
2016年4月 (2)
2016年2月 (1)
2016年1月 (2)
2015年6月 (1)
2015年4月 (1)
2015年2月 (2)
2015年1月 (1)
2014年12月 (3)
2014年7月 (1)
2014年6月 (3)
2014年5月 (1)
2014年4月 (1)
2014年3月 (3)
2014年2月 (1)
2014年1月 (4)
2013年12月 (2)



TraceView 界面比较复杂，其 UI 划分为上下两个面板，即 Timeline Panel（时间线面板）和 Profile Panel（分析面板）。上图中的上半部分为 Timeline Panel（时间线面板），Timeline Panel 又可细分为左右两个 Pane：

- 左边 Pane 显示的是测试数据中所采集的线程信息。由图可知，本次测试数据采集了 main 线程，传感器线程和其它系统辅助线程的信息。
- 右边 Pane 所示为时间线，时间线上是每个线程测试时间段内所涉及的函数调用信息。这些信息包括函数名、函数执行时间等。由图可知，Thread-1412 线程对应行的内容非常丰富，而其他线程在这段时间内干得工作则要少得多。

2013年11月 (3)
2013年10月 (1)
2013年8月 (3)
2013年7月 (6)
2013年6月 (1)
2013年5月 (6)
2013年4月 (6)
2013年3月 (14)
2013年2月 (15)
2013年1月 (19)

技术博客

AngelDevil
CoderRobin's Blog
Hongyang
Iceskysl
James Smith
JaveZh
Johnny901114
Kris on Mobile Development
OpenAphid-Engine
RayRay
scott's blog
Trinea
Way
xiaoQLu
youxiachai
短裤党
蓝斯的专栏
农民伯伯
谦虚的天下
任玉刚
小坦克
依旧淡然
应用开发笔记

- 另外，开发者可以在时间线 Pane 中移动时间线纵轴。纵轴上边将显示当前时间点中某线程正在执行的函数信息。

上图中的下半部分为 Profile Panel（分析面板），Profile Panel 是 TraceView 的核心界面，其内涵非常丰富。它主要展示了某个线程（先在 Timeline Panel 中选择线程）中各个函数调用的情况，包括 CPU 使用时间、调用次数等信息。而这些信息正是查找 hotspot 的关键依据。所以，对开发者而言，一定要了解 Profile Panel 中各列的含义。下表列出了 Profile Panel 中比较重要的列名及其描述。

列名	描述
Name	该线程运行过程中所调用的函数名
Incl Cpu Time	某函数占用的 CPU 时间，包含内部调用其它函数的 CPU 时间
Excl Cpu Time	某函数占用的 CPU 时间，但不含内部调用其它函数所占用的 CPU 时间
Incl Real Time	某函数运行的真实时间（以毫秒为单位），内含调用其它函数所占用的真实时间
Excl Real Time	某函数运行的真实时间（以毫秒为单位），不含调用其它函数所占用的真实时间
Call+Recur Calls/Total	某函数被调用次数以及递归调用占总调用次数的百分比
Cpu Time/Call	某函数调用 CPU 时间与调用次数的比，相当于该函数平均执行时间
Real Time/Call	同 CPU Time/Call 类似，只不过统计单位换成了真实时间

TraceView 实战

了解完 TraceView 的 UI 后，现在介绍如何利用 TraceView 来查找 hotspot。一般而言，hotspot 包括两种类型的函数：

- 一类是调用次数不多，但每次调用却需要花费很长时间的函数。
- 一类是那些自身占用时间不长，但调用却非常频繁的函数。

测试背景：APP 在测试机运行一段时间后出现手机发烫、卡顿、高 CPU 占有率的现象。将应用切入后台进行 CPU 数据的监测，结果显示，即使应用不进行任何操作，应用的 CPU 占有率都会持续的增长。

按照 TraceView 简介中的方法进行测试，TraceView 结果 UI 显示后进行数据分析，在 Profile Panel 中，选择按 Cpu Time/Call 进行降序排序（从上之下排列，每项的耗费时间由高到低）得到如图所示结果：

跃睿
技术社区

Android Arsenal
Android Design
Android 开发者
Code4App
Developer.com
FastJson
Git @ OSC
Github.com
Google Samples
Java开源大全
ShareSDK
Stack OverFlow
w3school 在线教程

Name	Cpu Time/Call	Incl Cpu Time %	Calls+RecurCalls/Total	Incl
2 com/cnki/client/utils/ImageLoaderTools\$2.run ()V	1111.124	31.1%	1+0	
8 android/app/ActivityThread.access\$600 (Landroid/app/ActivityThread;)Z	711.705	19.9%	1+0	
9 android/app/ActivityThread.handleLaunchActivity (Landroid/app/ActivityThread;)V	711.674	19.9%	1+0	
10 android/app/ActivityThread.performLaunchActivity (Landroid/app/ActivityThread;)V	698.490	19.6%	1+0	
12 android/app/Instrumentation.callActivityOnCreate (Landroid/app/Instrumentation;Landroid/app/ActivityThread;)V	691.164	19.3%	1+0	
13 android/app/Activity.performCreate (Landroid/os/Bundle;Landroid/app/ActivityThread;)V	691.134	19.3%	1+0	
14 com/cnki/client/act/C_Act_CatLog.onCreate (Landroid/os/Bundle;Landroid/app/ActivityThread;)V	691.043	19.3%	1+0	
22 com/cnki/client/act/C_Act_CatLog\$1.handleMessage (Landroid/os/Message;)V	390.233	10.9%	1+0	
7 android/app/ActivityThread\$H.handleMessage (Landroid/os/Message;)V	371.342	20.8%	2+0	
32 com/cnki/client/act/C_Act_CatLog.setupViews ()V	336.548	9.4%	1+0	

Find:

图中 ImageLoaderTools\$2.run() 是应用程序中的函数，它耗时为 1111.124。然后点击 ImageLoaderTools\$2.run() 项，得到更为详尽的调用关系图：

Name	Cpu Time/Call	Incl Cpu Time %	Calls+RecurCalls/Total	Incl
2 com/cnki/client/utils/ImageLoaderTools\$2.run ()V	1111.124	31.1%	1+0	
Parents				
0 (toplevel)		100.0%	1/1	
Children				
self		45.3%		
75 com/cnki/client/utils/ImageLoaderTools.access\$600 (Landroid/app/ActivityThread;)Z		14.4%	35221/35221	
67 java/util/ArrayList.size ()I		14.4%	35220/36321	
76 com/cnki/client/utils/ImageLoaderTools.access\$600 (Landroid/app/ActivityThread;)Z		14.3%	35220/35220	
151 com/cnki/client/utils/BitMapTools.saveBitmap (Landroid/graphics/Bitmap;Landroid/graphics/Bitmap;)V		4.7%	1/1	
152 com/cnki/client/utils/HttpTools.getStream (Ljava/lang/String;)Ljava/io/InputStream;		4.7%	1/1	

Find:

上图中 Parents 为 ImageLoaderTools\$2.run() 方法的调用者：Parents (the methods calling this method)；Children 为 ImageLoaderTools\$2.run() 调用的子函数或方法：Children (the methods called by this method)。本例中 ImageLoaderTools\$2.run() 方法的调用者为 Framework 部分，而 ImageLoaderTools\$2.run() 方法调用的自方法中我们发现有三个方法的 Incl Cpu Time % 占用均达到了 14% 以上，更离谱的是 Calls+RecurCalls/Total 显示这三个方法均被调用了 35000 次以上，从包名可以识别出这些方法为测试者自身所实现，由此可以判断 ImageLoaderTools\$2.run() 极有可能是手机发烫、卡顿、高 CPU 占用率的原因所在。

代码验证

大致可以判断是 ImageLoaderTools\$2.run() 方法出现了问题，下面找到这个方法进行代码上的验证：



```
1 package com.sunzn.app.utils;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.io.InputStream;
6 import java.lang.ref.SoftReference;
7 import java.util.ArrayList;
8 import java.util.HashMap;
9
10 import android.content.Context;
11 import android.graphics.Bitmap;
12 import android.os.Environment;
13 import android.os.Handler;
14 import android.os.Message;
15
16 public class ImageLoaderTools {
17
18     private HttpTools httptool;
19
20     private Context mContext;
21
22     private boolean isLoop = true;
23
24     private HashMap<String, SoftReference<Bitmap>> mHashMap_caches;
25
26     private ArrayList<ImageLoadTask> maArrayList_taskQueue;
27
28     private Handler mHandler = new Handler() {
29         public void handleMessage(android.os.Message msg) {
30             ImageLoadTask loadTask = (ImageLoadTask) msg.obj;
31             loadTask.callback.imageloaded(loadTask.path, loadTask.bitmap);
32         };
33     };
34 }
```

```
35     private Thread mThread = new Thread() {
36
37         public void run() {
38
39             while (isLoop) {
40
41                 while (maArrayList_taskQueue.size() > 0) {
42
43                     try {
44                         ImageLoadTask task = maArrayList_taskQueue.remove(0);
45
46                         if (Constant.LOADPICTYPE == 1) {
47                             byte[] bytes = httptool.getBytes(task.path, null,
HttpTools.METHOD_GET);
48                             task.bitmap = BitmapTools.getBitmap(bytes, 40, 40);
49                         } else if (Constant.LOADPICTYPE == 2) {
50                             InputStream in = httptool.getInputStream(task.path, null,
HttpTools.METHOD_GET);
51                             task.bitmap = BitmapTools.getBitmap(in, 1);
52                         }
53
54                         if (task.bitmap != null) {
55                             mHashMap_caches.put(task.path, new SoftReference<Bitmap>
(task.bitmap));
56                             File dir =
mContext.getExternalFilesDir(Environment.DIRECTORY_PICTURES);
57                             if (!dir.exists()) {
58                                 dir.mkdirs();
59                             }
60                             String[] path = task.path.split("/");
61                             String filename = path[path.length - 1];
62                             File file = new File(dir, filename);
63                             BitmapTools.saveBitmap(file.getAbsolutePath(), task.bitmap);
64                             Message msg = Message.obtain();
65                             msg.obj = task;
66                             mHandler.sendMessage(msg);
67                         }
68                     } catch (IOException e) {
69                         e.printStackTrace();

```

```
70         } catch (Exception e) {
71             e.printStackTrace();
72         }
73
74         synchronized (this) {
75             try {
76                 wait();
77             } catch (InterruptedException e) {
78                 e.printStackTrace();
79             }
80         }
81
82     }
83
84 }
85
86 };
87
88 };
89
90 public ImageLoaderTools(Context context) {
91     this.mContext = context;
92     httptool = new HttpTools(context);
93     mMap_caches = new HashMap<String, SoftReference<Bitmap>>();
94     mArrayList_taskQueue = new ArrayList<ImageLoaderTools.ImageLoadTask>();
95     mThread.start();
96 }
97
98 private class ImageLoadTask {
99     String path;
100     Bitmap bitmap;
101     Callback callback;
102 }
103
104 public interface Callback {
105     void imageloaded(String path, Bitmap bitmap);
106 }
107
108 public void quit() {
```



```
109         isLoop = false;
110     }
111
112     public Bitmap imageLoad(String path, Callback callback) {
113         Bitmap bitmap = null;
114         String[] path1 = path.split("/");
115         String filename = path1[path1.length - 1];
116
117         if (mHashMap_caches.containsKey(path)) {
118             bitmap = mHashMap_caches.get(path).get();
119             if (bitmap == null) {
120                 mHashMap_caches.remove(path);
121             } else {
122                 return bitmap;
123             }
124         }
125
126         File dir = mContext.getExternalFilesDir(Environment.DIRECTORY_PICTURES);
127
128         File file = new File(dir, filename);
129
130         bitmap = BitMapTools.getBitMap(file.getAbsolutePath());
131         if (bitmap != null) {
132             return bitmap;
133         }
134
135         ImageLoadTask task = new ImageLoadTask();
136         task.path = path;
137         task.callback = callback;
138         maArrayList_taskQueue.add(task);
139
140         synchronized (mThread) {
141             mThread.notify();
142         }
143
144         return null;
145     }
146
147 }
```



以上代码即是 ImageLoaderTools 图片工具类的全部代码，先不着急去研究这个类的代码实现过程，先来看看这个类是怎么被调用的：



```
1 ImageLoaderTools imageLoaderTools = imageLoaderTools = new ImageLoaderTools(this);
2
3 Bitmap bitmap = imageLoaderTools.imageLoad(picpath, new Callback() {
4
5     @Override
6     public void imageloaded(String picPath, Bitmap bitmap) {
7         if (bitmap == null) {
8             imageView.setImageResource(R.drawable.default);
9         } else {
10             imageView.setImageBitmap(bitmap);
11         }
12     }
13 });
14
15 if (bitmap == null) {
16     imageView.setImageResource(R.drawable.fengmianmoren);
17 } else {
18     imageView.setImageBitmap(bitmap);
19 }
```



ImageLoaderTools 被调用的过程非常简单：1.ImageLoaderTools 实例化；2.执行 imageLoad() 方法加载图片。

在 ImageLoaderTools 类的构造函数（90行-96行）进行实例化过程中完成了网络工具 HttpTools 初始化、新建一个图片缓存 Map、新建一个下载队列、开启下载线程的操作。这时候请注意开启线程的操作，开启线程后执行 run() 方法（35行-88行），这时 isLoop 的值是默认 true，maArrayList_taskQueue.size() 是为 0 的，在任务队列 maArrayList_taskQueue 中还没有加入下载任务之前这个循环会一直循环下去。在执行 imageLoad() 方法加载图片时会首先去缓存 mHashMap_caches 中查找该图片是否已经被下载过，如果已经下载过则直接返回与之对应的 bitmap 资源，如果没有查找到则会往 maArrayList_taskQueue 中添加下载任务并唤醒对应的下载线程，之前开启的线程在发现 maArrayList_taskQueue.size() > 0 后就进入下载逻辑，下载完任务完成后将对应的图片资源加入缓存 mHashMap_caches 并更新 UI，下载线程执行 wait() 方法被挂起。一个图片下载的业务逻辑这样理解起来很顺

畅，似乎没有什么问题。开始我也这样认为，但后来在仔细的分析代码的过程中发现如果同样一张图片资源重新被加载就会出现死循环。还记得缓存 mHashMap_caches 么？如果一张图片之前被下载过，那么缓存中就会有这张图片的引用存在。重新去加载这张图片的时候如果重复的去初始化 ImageLoaderTools，线程会被开启，而使用 imageLoad() 方法加载图片时发现缓存中存在这个图片资源，则会将其直接返回，注意这里使用的是 `return bitmap`；那就意味着 imageLoad() 方法里添加下载任务到下载队列的代码不会被执行到，这时候 run() 方法中的 isLoop = true 并且 maArrayList_taskQueue.size() = 0，这样内层 while 里的逻辑也就是挂起线程的关键代码 `wait()` 永远不会被执行到，而外层 while 的判断条件一直为 true，就这样程序出现了死循环。死循环才是手机发烫、卡顿、高 CPU 占用率的真正原因所在。

解决方案

准确的定位到代码问题所在后，提出解决方案就很简单了，这里提供的解决方案是将 wait() 方法从内层 while 循环提到外层 while 循环中，这样重复加载同一张图片时，死循环一出现线程就被挂起，这样就可以避免死循环的出现。代码如下：

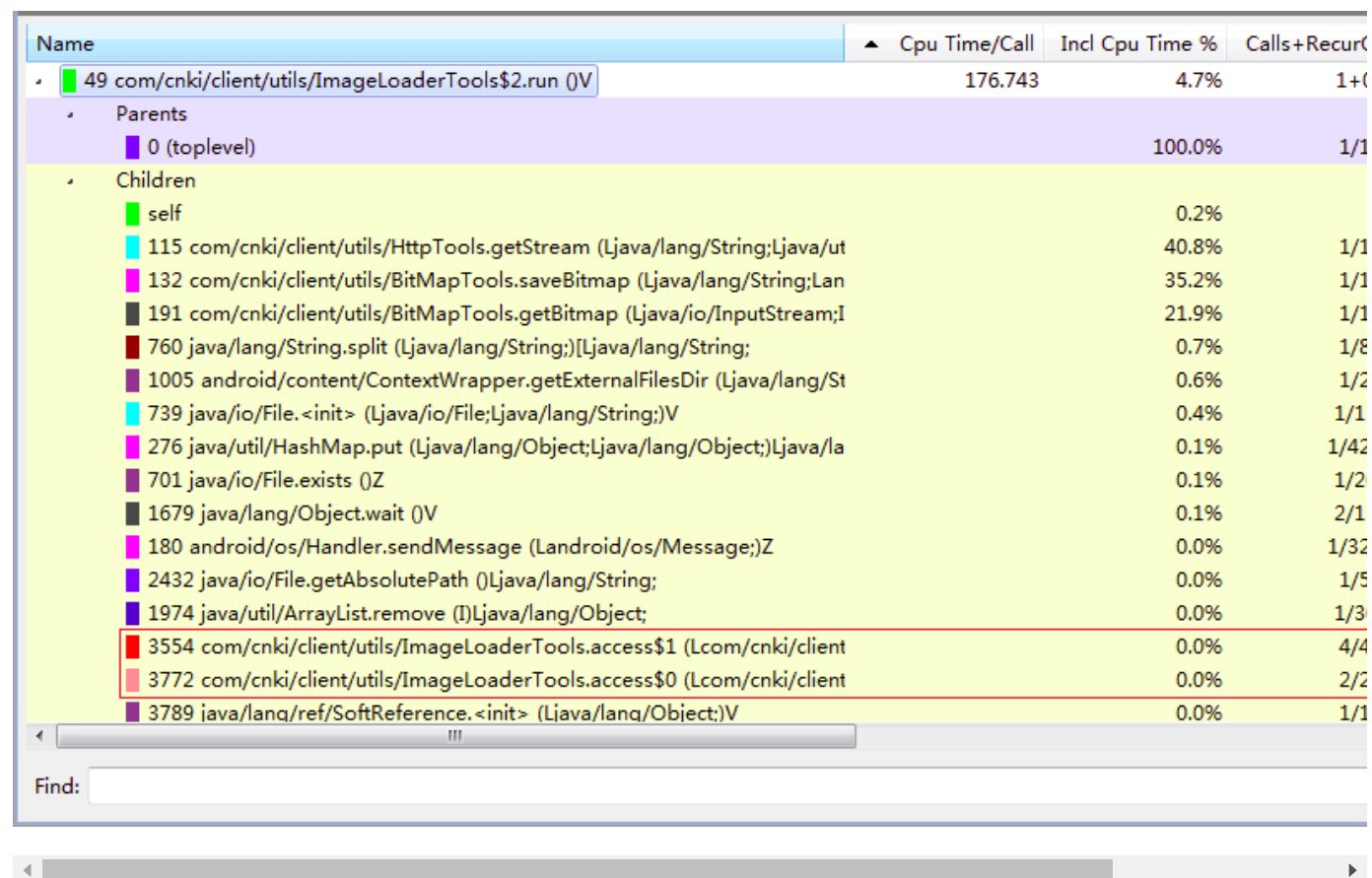
```

1 private Thread mThread = new Thread() {
2
3     public void run() {
4
5         while (isLoop) {
6
7             while (maArrayList_taskQueue.size() > 0) {
8
9                 try {
10                     ImageLoadTask task = maArrayList_taskQueue.remove(0);
11
12                     if (Constant.LOADPICTYPE == 1) {
13                         byte[] bytes = httpptool.getBytes(task.path, null,
14 HttpTools.METHOD_GET);
15                         task.bitmap = BitMapTools.getBitmap(bytes, 40, 40);
16                     } else if (Constant.LOADPICTYPE == 2) {
17                         InputStream in = httpptool.getInputStream(task.path, null,
18 HttpTools.METHOD_GET);
19                         task.bitmap = BitMapTools.getBitmap(in, 1);
20                     }
21
22                     if (task.bitmap != null) {
```

```
21         mHashMap_caches.put(task.path, new SoftReference<Bitmap>
(task.bitmap));
22         File dir =
mContext.getExternalFilesDir(Environment.DIRECTORY_PICTURES);
23         if (!dir.exists()) {
24             dir.mkdirs();
25         }
26         String[] path = task.path.split("/");
27         String filename = path[path.length - 1];
28         File file = new File(dir, filename);
29         BitmapTools.saveBitmap(file.getAbsolutePath(), task.bitmap);
30         Message msg = Message.obtain();
31         msg.obj = task;
32         mHandler.sendMessage(msg);
33     }
34     } catch (IOException e) {
35         e.printStackTrace();
36     } catch (Exception e) {
37         e.printStackTrace();
38     }
39
40 }
41
42 synchronized (this) {
43     try {
44         wait();
45     } catch (InterruptedException e) {
46         e.printStackTrace();
47     }
48 }
49
50 }
51
52 };
53
54 };
```



最后再附上代码修改后代码运行的性能图，和之前的多次被重复执行，效率有了质的提升，手机发烫、卡顿、高 CPU 占用率的现象也消失了。



专注移动互联网产品设计研发 分享最新的移动互联网产品和技术

分类: [安卓开发](#), [应用技巧](#)

标签: [调试](#)

好文要顶

关注我

收藏该文





sunzn

关注 - 0

粉丝 - 127

[+加关注](#)

4

0

[« 上一篇：Android 编程下的日志工具类](#)[» 下一篇：Android 编程下 java.lang.NoClassDefFoundError: cn.jpusth.android.api.JPushInterface 报错](#)

posted @ 2013-07-22 22:06 sunzn 阅读(33738) 评论(5) 编辑 收藏

评论列表

#1楼

2015-12-10 14:20 特立独行A

重新去加载这张图片的时候如果重复的去初始化 ImageLoaderTools，线程会被开启，而使用 imageLoad() 方法加载图片时发现缓存中存在这个图片资源，则会将其直接返回，注意这里使用的是 return bitmap; 那就意味着 imageLoad() 方法里添加下载任务到下载队列的代码不会被执行到，这时候 run() 方法中的 isLoop = true 并且 maArrayList_taskQueue.size() = 0.

我想问你 重新初始化了 ImageLoaderTools，你的 mHashMap_caches这个类变量不是重新分配吗，你是不是哪里搞错了，问题的关键你估计是找错了吧。

我认为问题的关键就是，isLoop = true，如果 一直不往队列里面添加任务，那么这里 外层while循环一直执行，一直在消耗cpu，而不是楼主说的那个mHashMap_caches的使用问题，mHashMap_caches的使用没有问题，类的对象的构造你可能没搞懂，类所有成员都是跟类对象一起的，除了静态变量这些是类持有，其他都是对象持有。

支持(2) 反对(0)

#2楼

[\[楼主\]](#) 2015-12-10 15:50 sunzn[@ 特立独行A](#)

重新初始化 ImageLoaderTools，mHashMap_caches这个类变量重新分配，请注意构造方法，这时候空转的线程已经开启，这时候while (isLoop)内层的代码不会被执行到，这时候线程在等待下载任务的加入。ImageLoaderTools.imageLoad方法被调用的时候，bitmap = BitmapTools.getBitmap(file.getAbsolutePath()); 先去获取缓存，如果缓存存在，获取到缓存后，直接返回Bitmap，后续maArrayList_taskQueue.add(task);没有被执行到，而之前开启的线程一直在等待任务加入，造成死循环。有质疑是好事，请仔细查看代码，想明白为什么要将同步代码提到while (maArrayList_taskQueue.size() > 0)外层。

[支持\(0\)](#) [反对\(0\)](#)

#3楼

2016-03-10 15:43 苏晓晓

楼主写的挺详细的，如果能多举点时例给我们小菜鸟参考一下就完美哒[支持\(0\)](#) [反对\(0\)](#)

#4楼

2016-03-31 14:51 儒雅小生

楼主很细心，分析的很到位！不过我感觉1楼说得有道理，先不说缓存里存不存在图片资源，isLoop = true 所以mThread 线程一开启就陷入了死循环，说的不对请指教[支持\(0\)](#) [反对\(0\)](#)

#5楼

2016-12-21 19:58 I_am_coder

2楼楼主的回复并没有明确指出1楼分析不对的地方，楼主可否把完整的代码贴出来供大家仔细分析，我也没看明白楼主的分析逻辑。[支持\(0\)](#) [反对\(0\)](#)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【推荐】腾讯云上实验室 1小时搭建人工智能应用

【推荐】可嵌入您系统的“在线Excel”！SpreadJS 纯前端表格控件

【推荐】阿里云“全民云计算”优惠升级



最新IT新闻:

- 东芝向财团出售芯片交易推迟 因苹果不同意部分关键条款
 - 想要实现自动驾驶？高精度地图不可或缺
 - 智联招聘将退市美股，股东已批准合并协议
 - 腾讯官方回应微信卡死Bug：正在紧急修复
 - 微软推出三款新机器学习工具 帮助开发者打造AI应用
- » 更多新闻...



最新知识库文章:

- 如何阅读计算机科学类的书
 - Google 及其云智慧
 - 做到这一点，你也可以成为优秀的程序员
 - 写给立志做码农的大学生
 - 架构腐化之谜
- » 更多知识库文章...

