

# Android 应用性能优化经验分享

# 关于我 about me

张明云



就职于  
步步高教育电子



4年  
Android 应用开发经验



技术管理  
软件工程  
组件化  
应用性能优化



知乎  
编程、Android开发话题  
优秀回答者



简书专题  
《Android开发经验谈》  
运营者



微信公众号：open\_dev

# 目录

## CONTENT

1. 性能优化有多重要？
2. 应用性能状况
3. 性能优化流程、原则、指标和工具
4. 性能优化实战
5. 实际优化效果

# 目录

## CONTENT

1. 性能优化有多重要？
2. 应用性能状况
3. 性能优化流程、原则、指标和工具
4. 性能优化实战
5. 实际优化效果

# 01 性能优化有多重要？

作为手机用户，每天都有哪些问题让我们烦恼？

手机莫名发烫

手机越用越卡

掉电快，充满一次电还管不了一天

手机屏幕上密密麻麻的指纹印，难看、难擦

网络问题，无内容显示/持续转圈

# 01 性能优化有多重要？

做应用的挑战不只是需要把工作做完，而是要做得足够出色

改善性能就是改善产品体验

同类竞品越来越多，优胜劣汰

用户越来越“挑”

# 01 性能优化有多重要？



# 目录

## CONTENT

1. 性能优化有多重要？
2. 应用性能状况
3. 性能优化流程、原则、指标和工具
4. 性能优化实战
5. 实际优化效果



## 02 应用性能状况

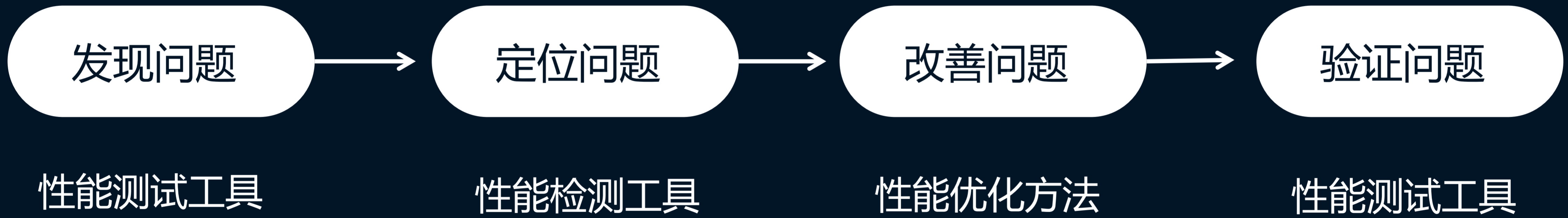
- 约60%应用冷启动时间超过2S
- SDK的不合理使用（基础类型和装箱类型、HashMap和SparseArray）
- 在系统回调或频繁调用的代码块中创建新的实例
- 几乎所有的APP都存在过渡绘制问题，Activity和Window都设置了背景
- json库的不合理使用，导致Launcher严重卡顿
- 近10个应用监听开机广播，导致开机后一段时间内Launcher严重卡顿
- 应用内存占用不合理（适配不规范、缓存不合理、回收不及时）
- 系统SDK导致的内存泄漏（InputMethodManager、WebView，[AndroidExcludedRefs.java](#)）
- 非静态内部类导致的内存泄漏（Handler、Observer、AsyncTask）
- 四大组件的Context和Application Context的不合理使用
- IO操作完成后没有关闭文件（Cursor、TypedArray、File等）
- 功耗问题明显（循环动画、过渡绘制、网络请求不合理、后台服务常驻等）

# 目录

## CONTENT

1. 性能优化有多重要？
2. 应用性能状况
3. 性能优化流程、原则、指标和工具
4. 性能优化实战
5. 实际优化效果

# 03 性能优化流程



## 03 性能优化原则

- 不能凭感觉，要看数据说话，有足够多的测量
- 尽量使用低配置设备进行测试
- 权衡利弊，以保证进度、稳定为主
- 改善后一定要验证，保证每一次改善都有效，不会导致其它问题

# 03 性能优化指标

指标	现象	原因	检测工具
启动速度	黑、白屏 首帧出现慢	布局复杂 UI阻塞 过渡绘制 初始化做的事情太多	Hierarchy Viewer StrictMode TraceView Hugo
流畅度	应用卡顿	内存抖动 过渡绘制 UI阻塞 整机剩余内存不足	Android Studio 开发者选项 StrictMode TraceView Hugo
内存	整机慢	内存占用多 内存抖动 内存泄漏	Android Studio Leakcanary
功耗	发烫 掉电快	硬件持续高负荷工作 频繁的网络访问 频繁的界面刷新 长时间不待机 后台持续工作	Android Studio 开发者选项

# 03 性能优化指标

指标	现象	原因	检测工具
启动速度	黑、白屏 首帧出现慢	布局复杂 UI阻塞 过渡绘制 初始化做的事情太多	Hierarchy Viewer StrictMode TraceView Hugo
流畅度	应用卡顿	内存抖动 过渡绘制 UI阻塞 整机剩余内存不足	Android Studio 开发者选项 StrictMode TraceView Hugo
内存	整机慢	内存占用多 内存抖动 内存泄漏	Android Studio Leakcanary
功耗	发烫 掉电快	硬件持续高负荷工作 频繁的网络访问 频繁的界面刷新 长时间不待机 后台持续工作	Android Studio 开发者选项

# 03 性能优化指标

指标	现象	原因	检测工具
启动速度	黑、白屏 首帧出现慢	布局复杂 UI阻塞 过渡绘制 初始化做的事情太多	Hierarchy Viewer StrictMode TraceView Hugo
流畅度	应用卡顿	内存抖动 过渡绘制 UI阻塞 整机剩余内存不足	Android Studio 开发者选项 StrictMode TraceView Hugo
内存	整机慢	内存占用多 内存抖动 内存泄漏	Android Studio Leakcanary
功耗	发烫 掉电快	硬件持续高负荷工作 频繁的网络访问 频繁的界面刷新 长时间不待机 后台持续工作	Android Studio 开发者选项

# 03 性能优化指标

指标	现象	原因	检测工具
启动速度	黑、白屏 首帧出现慢	布局复杂 UI阻塞 过渡绘制 初始化做的事情太多	Hierarchy Viewer StrictMode TraceView Hugo
流畅度	应用卡顿	内存抖动 过渡绘制 UI阻塞 整机剩余内存不足	Android Studio 开发者选项 StrictMode TraceView Hugo
内存	整机慢	内存占用多 内存抖动 内存泄漏	Android Studio Leakcanary
功耗	发烫 掉电快	硬件持续高负荷工作 频繁的网络访问 频繁的界面刷新 长时间不待机 后台持续工作	Android Studio 开发者选项



# 03 性能优化工具 ( AS Inspect Code )

The screenshot displays the Android Studio interface. The main editor shows the `AppInfoDBManager.java` file with the following code:

```
43  
44 Cursor cursor = sqLiteDatabase.query(AppInfoDBHelper.TABLE_NAME, projections, selection, selectionArgs, null, null,  
45 if(null != cursor && cursor.getCount()>0){  
46     cursor.moveToFirst();  
47     do{  
48         packageName = cursor.getString(cursor.getColumnIndex(AppInfoDBHelper.KEY_PACKAGE_NAME));  
49         if(AppInfoManager.isAppInstalled(context, packageName)){  
50             // 安装了才显示  
51             AppInfo appInfo = new AppInfo();  
52             appInfo.setAppName(cursor.getString(cursor.getColumnIndex(AppInfoDBHelper.KEY_APP_NAME)));  
53             appInfo.setPackageName(packageName);  
54             appInfo.setDrawableIcon(BitmapUtils.bytes2Bitmap(cursor.getBlob(cursor.getColumnIndex(AppInfoDBHelper.KEY_APP_ICON))));  
55             appInfo.setAppType(cursor.getString(cursor.getColumnIndex(AppInfoDBHelper.KEY_APP_TYPE)));  
56             appInfo.setEnableUnInstall(cursor.getInt(cursor.getColumnIndex(AppInfoDBHelper.KEY_SYSTEM_APP))==1);  
57  
58             appInfoList.add(appInfo);  
59         }  
60     }while(cursor.moveToNext());  
61     cursor.close();  
62 }  
63  
64 sqLiteDatabase.close();  
65  
66 return appInfoList;  
67
```

The AS Inspect Code tool results are shown in the bottom-left pane, listing various inspection categories:

- PreSchoolWidget (514 items)
- Android (41 items)
- Android > Lint > Correctness (5 items)
- Android > Lint > Internationalization (1 item)
- Android > Lint > Internationalization > Bidirectional Text (2 items)
- Android > Lint > Security (1 item)
- Android > Lint > Usability (1 item)
- Android > Lint > Usability > Icons (3 items)
- Code maturity issues (11 items)
- Compiler issues (1 item)
- Data flow issues (2 items)
- Declaration redundancy (137 items)
- General (10 items)
- Imports (1 item)
- Javadoc issues (57 items)
- Probable bugs (4 items)
- Properties Files (1 item)
- Spelling (109 items)
- Verbose or redundant code constructs (2 items)
- XML (125 items)

The bottom-right pane displays the text: "改善后再次检测，报的问题消失了。" (After improvement, re-inspection, the reported issues disappeared.)

# 03 性能优化工具 (AS Performance Monitor)

The screenshot displays the Android Studio interface with the Performance Monitor open. The top toolbar shows the 'AppInfoDBManager.java' file. The Performance Monitor is divided into four sections: Memory, CPU, GPU, and Network. Each section has a title bar with a monitor icon, a name, and a status bar with control buttons. The Memory monitor shows a bar chart of memory usage over time, with a legend indicating 'Free [1.19 MB]' and 'Allocated [36.75 MB]'. The CPU monitor shows a line graph of CPU usage percentage, with a legend indicating 'User [2.35 %]' and 'Kernel [0.00 %]'. The GPU monitor shows a line graph of GPU execution time in milliseconds, with a legend listing various categories like 'Misc Time', 'Swap Buffers', 'Command Issue', 'Sync', 'Draw', 'Measure/Layout', 'Animation', 'Input Handling', and 'VSync Delay'. The Network monitor shows a line graph of network data transfer rates in KB/s, with a legend indicating 'Tx [0.00 KB/s]' and 'Rx [0.00 KB/s]'. The status bar at the bottom shows 'Gradle build finished in 52s 769ms (15 minutes ago)' and the system time '64:32'.

- 1、选择连接到PC上的Android设备;
- 2、选择当前设备中某一个正在运行的进程;
- 3、Memory Monitor, 从左右到的按钮分别是:  
Enabled: 控制Memory Monitor是否处于工作状态;  
Initiate GC: 手动触发GC, 模拟虚拟机的GC操作;  
Dump Java Heap: 获取该进程当前状态下的内存快照;  
Start Allocation Tracking: 获取该进程某一段时间内的内存申请情况
- 4、当前进程实时的内存申请和释放情况;
- 5、CPU Monitor, 从左到右的按钮分别是:  
Enabled: 同Memory Monitor  
Start Method Tracing: 获取该进程某一段时间内各个执行方法的耗时情况;
- 6、GPU Monitor, 从左到右的按钮分别是:  
Enabled: 同Memory Monitor  
第二个按钮: GPU调试工具, 需要安装后才能使用;
- 7、将每一帧的执行耗时拆分成9种类型, GPU工作时, 会实时显示每一种类型的执行耗时情况;
- 8、Network Monitor  
Enabled: 同Memory Monitor

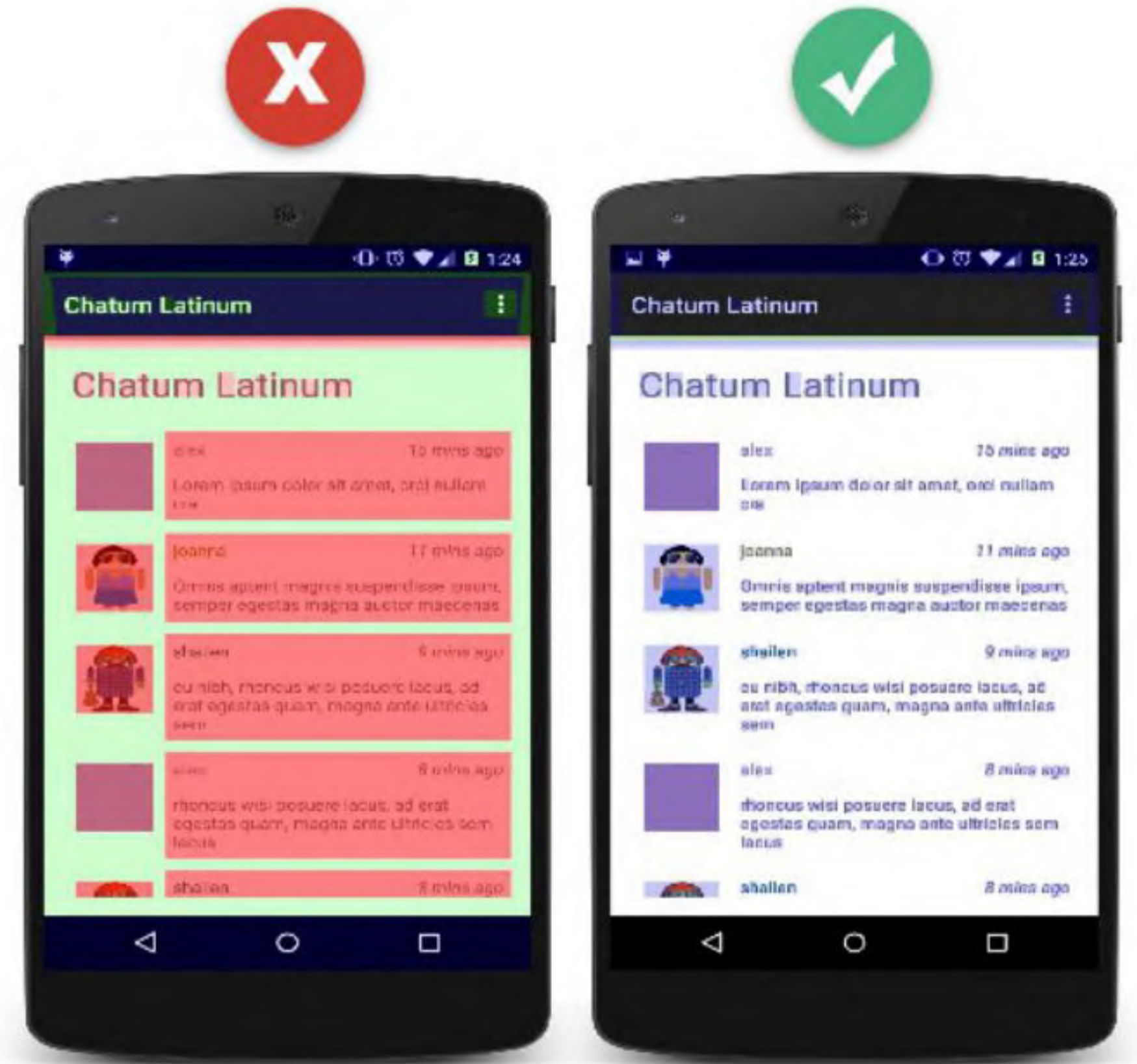
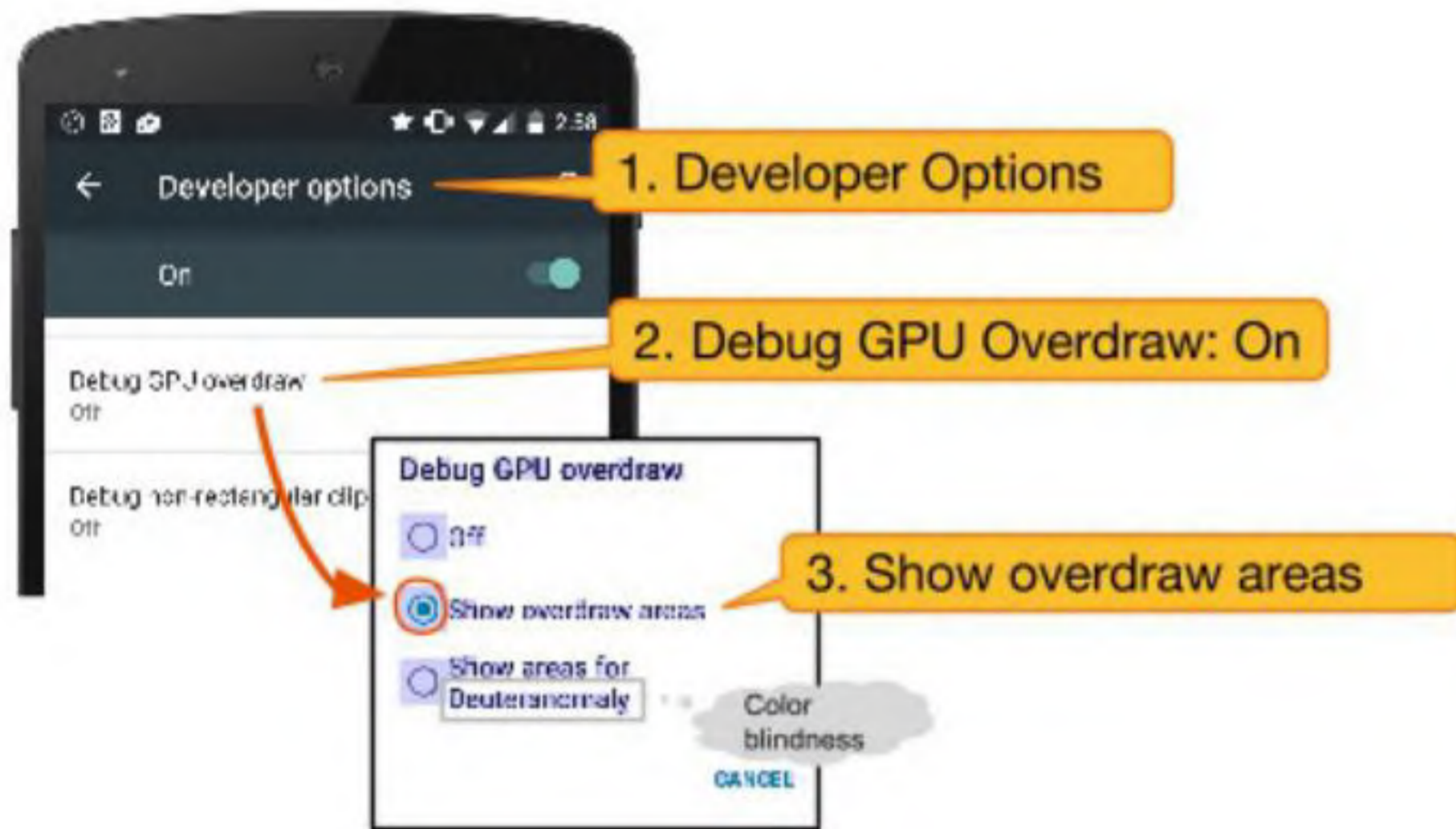
# 03 性能优化工具 (AS Data Analysis)

28 Java Heap: 21120  
29 Native Heap: 27608  
30 Code: 27304  
31 Stack: 748  
32 Graphics: 13760  
33 Private Other: 6084  
34 System: 10062  
35  
36 TOTAL: 106686 TOTAL SWAP (KB): 0  
37  
38 Objects  
39 Views: 448 ViewRootImpl: 0  
40 AppContexts: 5 4 Activities: 2  
41 Assets: 4 AssetManagers: 2  
42 Local Binders: 28 Proxy Binders: 25  
43 Parcel memory: 16 Parcel count: 57  
44 Death Recipients: 1 OpenSSL Sockets: 0  
45  
46 SQL

1、把程序的所有功能都玩一遍，退出程序后手动触发GC；  
2、可以看到Memory Monitor中申请的内存有明显的下降，因为回收了不再使用的内存；  
3、点击3中的Memory Usage选项，它相当于执行命令行“adb shell dumpsys meminfo packagename”，用于查看该进程的当前内存和组件状态；  
4、执行3后，会生成一个后缀为.txt的文件，里面包括该进程当前状态下各种类型的内存占用情况，以及组件的个数，完全退出程序后，无内存泄露的情况下Views和Activities的数量应该为0，AppContexts的数量应该为1，即Application Context。

Memory Usage工具主要用于查看某个进程在某个状态下的内存情况，以及大致定位内存泄露的位置。

# 03 性能优化工具（开发者选项 过渡绘制）



# 03 性能优化工具（开发者选项 GPU呈现模式分析）



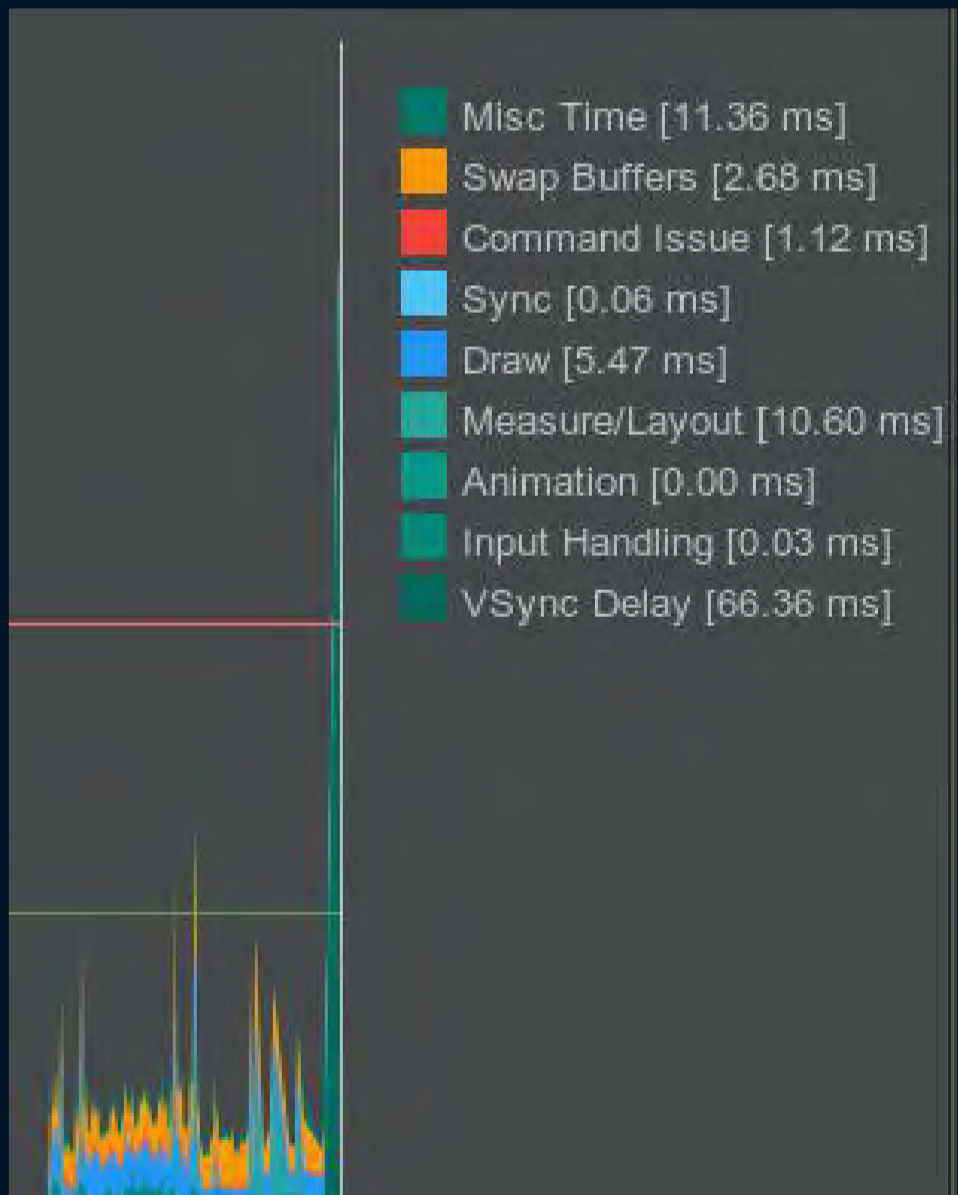
1. Settings > Developer Options

2. Tap to turn Profile GPU Rendering on

3. Choose On screen as bars



API < 23



API >= 23

# 03 性能优化工具 (StrictMode)

```
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()  
    .detectDiskReads()  
    .detectDiskWrites()  
    .detectNetwork() // or .detectAll() for all detectable problems  
    .penaltyLog()  
    .build());  
StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()  
    .detectLeakedSqlLiteObjects()  
    .detectLeakedClosableObjects()  
    .penaltyLog()  
    .penaltyDeath()  
    .build());
```



# 03 性能优化工具 (TraceView)

左侧时间线面板：左侧显示每一个工作线程，右侧用不同颜色标明每一个工作线程下的每个方法的执行耗时，横向宽度越宽表示越耗时。

中间分析面板：  
**Name:** 方法名，点击展开某一个方法包括Parent、Children两部分，Parent表示调用这个方法的方法，即其父方法；Children表示这个方法中调用的其它方法，即其子方法。  
**Incl Cpu Time:** 表示某个方法执行的总时间，上图中可以看到0 (toplevel) 的Incl Cpu Time 占了100%的时间，这个不是说100%的时间都是它在执行，请看下面代码：

```
public void top() {
    a();
    b();
    c();
    d();
}
```

**Incl Cpu Time**表示方法top执行的总时间，假如说方法top的执行时间为10ms，方法a执行了1ms，方法b执行了2ms，方法c执行了3ms，方法d执行了4ms（这里是为了举个栗子，实际情况中方法a、b、c、d的执行总时间肯定比方法top的执行总时间要小一点）。而且调用方法top的方法的执行时间是100ms，那么：

- top 10%
- a 10%
- b 20%
- c 30%
- d 40%

**Excl Cpu Time:** 方法top的Incl Cpu Time减去方法a、b、c、d的Incl Cpu Time的时间就是方法top的Excl Cpu Time了

**Calls + Recur Calls/Total:** 它表示这个方法调用的次数，这个指标越高越好，Recalls表示这个方法调用的次数。

底部关键词过滤面板

Name	Incl Cpu Time %	Incl Cpu Time	Excl Cpu Time %	Excl Cpu Time	Incl Real Time %	Incl Real Time	Excl Real Time %	Excl Real Time	Calls+RecurCalls/Total	Cpu Time/Call	Real Time/Call
0 (toplevel)	100.0%	2071.114	0.1%	1.174	100.0%	13055.919	0.0%	0.000	12+0	172.593	1087.993
1 com/zhangmy/preschoolwidget/ui/activity/HomeActivity.initHome (JV)	78.9%	1634.197	0.0%	0.058	25.2%	3283.825	0.0%	0.059	1+0	1634.197	3283.825
2 com/zhangmy/preschoolwidget/ui/activity/HomeActivity.addWidget2Launcher (JV)	40.9%	846.242	0.0%	0.390	15.2%	1990.966	0.0%	0.394	1+0	846.242	1990.966
3 com/zhangmy/preschoolwidget/widget/AutoAddAppWidgetHelper.autoAddAppWidget (Landroid/con	40.8%	845.133	0.0%	0.158	15.2%	1989.858	0.0%	0.161	1+0	845.133	1989.858
4 com/zhangmy/preschoolwidget/widget/AutoAddAppWidgetHelper.autoAddAppWidgetToNewLaunche	40.3%	835.329	0.0%	0.468	14.9%	1939.259	0.0%	0.480	1+0	835.329	1939.259
5 com/zhangmy/preschoolwidget/ui/activity/HomeActivity.blurBackground (JV)	33.7%	698.585	0.0%	0.483	9.1%	1191.201	0.0%	0.499	1+0	698.585	1191.201
6 com/zhangmy/preschoolwidget/widget/AutoAddAppWidgetHelper.isNewLauncherSupportAutoAdd (L	33.3%	690.050	14.3%	295.257	9.0%	1177.455	2.3%	294.775	1+0	690.050	1177.455
7 java/lang/System.currentTimeMillis (J)	28.2%	583.762	28.2%	583.762	8.6%	1118.639	4.4%	580.880	200694+0	0.003	0.006
Parents											
6 com/zhangmy/preschoolwidget/widget/AutoAddAppWidgetHelper.isNewLauncherSupportAuto/	65.8%	383.854			51.3%	574.292			136525/200694		
20 com/zhangmy/preschoolwidget/widget/AutoAddAppWidgetHelper\$CheckAddStateTask.doInB	34.2%	199.908			48.7%	544.347			64169/200694		
Children											
self	100.0%	583.762			51.9%	580.880			359/1173		
(context switch)	0.0%	0.000			48.1%	537.759					
8 android/app/WallpaperManager.getDrawable (Landroid/graphics/drawable/Drawable;	21.7%	449.685	0.0%	0.046	5.5%	718.038	0.0%	0.050	1+0	449.685	718.038
9 android/app/WallpaperManager\$Globals.peekWallpaperBitmap (Landroid/content/Context;Z)Landroic	21.7%	449.188	0.0%	0.013	5.5%	716.606	0.0%	0.014	1+0	449.188	716.606
10 android/app/WallpaperManager\$Globals.getCurrentWallpaperLocked (Landroid/content/Context;L	21.7%	449.175	0.0%	0.097	5.5%	716.592	0.0%	0.101	1+0	449.175	716.592
11 android/graphics/BitmapFactory.decodeFileDescriptor (Ljava/io/FileDescriptor;Landroid/graphics/Rec	21.5%	445.971	0.0%	0.085	5.4%	706.576	0.0%	0.046	1+0	445.971	706.576
12 android/graphics/BitmapFactory.nativeDecodeFileDescriptor (Ljava/io/FileDescriptor;Landroi	21.5%	445.822	20.9%	433.506	5.4%	704.195	3.3%	433.502	1+0	445.822	704.195
13 java/lang/Thread.run (JV)	16.7%	346.645	0.0%	0.011	9.9%	1292.871	0.0%	0.008	1+0	346.645	1292.871
14 java/util/concurrent/ThreadPoolExecutor\$Worker.run (JV)	16.7%	346.634	0.0%	0.004	9.9%	1292.863	0.0%	0.004	1+0	346.634	1292.863
15 java/util/concurrent/ThreadPoolExecutor.runWorker (Ljava/util/concurrent/ThreadPoolExecuto	16.7%	346.630	0.0%	0.030	9.9%	1292.859	0.0%	0.030	1+0	346.630	1292.859
16 android/os/AsyncTask\$SerialExecutor\$1.run (JV)	16.7%	346.435	0.0%	0.017	9.9%	1292.664	0.0%	0.017	1+0	346.435	1292.664
17 java/util/concurrent/FutureTask.run (JV)	16.7%	346.418	0.0%	0.063	9.9%	1292.647	0.0%	0.063	1+0	346.418	1292.647

## 03 性能优化工具（方法耗时打印 Hugo）

```
buildscript {
    repositories {
        mavenCentral()
    }

    dependencies {
        classpath 'com.jakewharton.hugo:hugo-plugin:1.2.1'
    }
}

apply plugin: 'com.android.application'
apply plugin: 'com.jakewharton.hugo'
```

Disable logging temporarily by adding the following:

```
hugo {
    enabled false
}
```

If you want to toggle logging at runtime, use `Hugo.setEnabled(true|false)`

```
@DebugLog
public String getName(String first, String last) {
    SystemClock.sleep(15); // Don't ever really do this!
    return first + " " + last;
}
```

```
V/Example: → getName(first="Jake", last="Wharton")
```

```
V/Example: ← getName [16ms] = "Jake Wharton"
```



# 03 性能优化工具（布局性能查看 Hierarchy Viewer）

The screenshot displays the Hierarchy Viewer interface. The main area shows a tree of views:

- PhoneWindow\$DecorView (@40518230) contains a LinearLayout (@40518a70).
- The LinearLayout (@40518a70) contains a FrameLayout (@40519470) and another LinearLayout (@4051a800).
- The FrameLayout (@40519470) contains a TextView (@40519a60).
- The LinearLayout (@4051a800) contains a FrameLayout (@4051ad00).
- The FrameLayout (@4051ad00) contains a TextView (@4051af68) and another FrameLayout (@4051b490).
- The FrameLayout (@4051b490) contains an ImageView (@4051bee0) which displays a fish image.

Performance metrics for the fish image's ImageView are shown in a tooltip:

- Measure: 0.066 ms
- Layout: 0.040 ms
- Draw: 8.454 ms

On the right, a property table is visible:

Property	Value
getFilterTouchesW	false
getTag()	null
getVisibility()	VISIBLE
isClickable()	false
isEnabled()	true
isFocusableInTouch	false
isHapticFeedbackEi	true
isInTouchMode()	true
isSelected()	false
isSoundEffectsEnal	true
mID	id/imageView_Fish
mPrivateFlags	16911408
mPrivateFlags_DRA	0x0
mPrivateFlags_DRA	0x20
mViewFlags	402653184

At the bottom right, a visual representation of the view hierarchy is shown with a red box highlighting the fish image.

# 03 性能优化 ( 内存泄露检测 Leakcanary )

## How do I use it?

Use a `RefWatcher` to watch references that should be GCed:

```
RefWatcher refWatcher = {...};

// We expect schrodingerCat to be gone soon (or not), let's watch it.
refWatcher.watch(schrodingerCat);
```

`LeakCanary.install()` returns a pre configured `RefWatcher`. It also installs an `ActivityRefWatcher` that automatically detects if an activity is leaking after `Activity.onDestroy()` has been called.

```
public class ExampleApplication extends Application {

    public static RefWatcher getRefWatcher(Context context) {
        ExampleApplication application = (ExampleApplication) context.getApplicationContext();
        return application.refWatcher;
    }

    private RefWatcher refWatcher;

    @Override public void onCreate() {
        super.onCreate();
        refWatcher = LeakCanary.install(this);
    }
}
```

You could use the `RefWatcher` to watch for fragment leaks:

```
public abstract class BaseFragment extends Fragment {

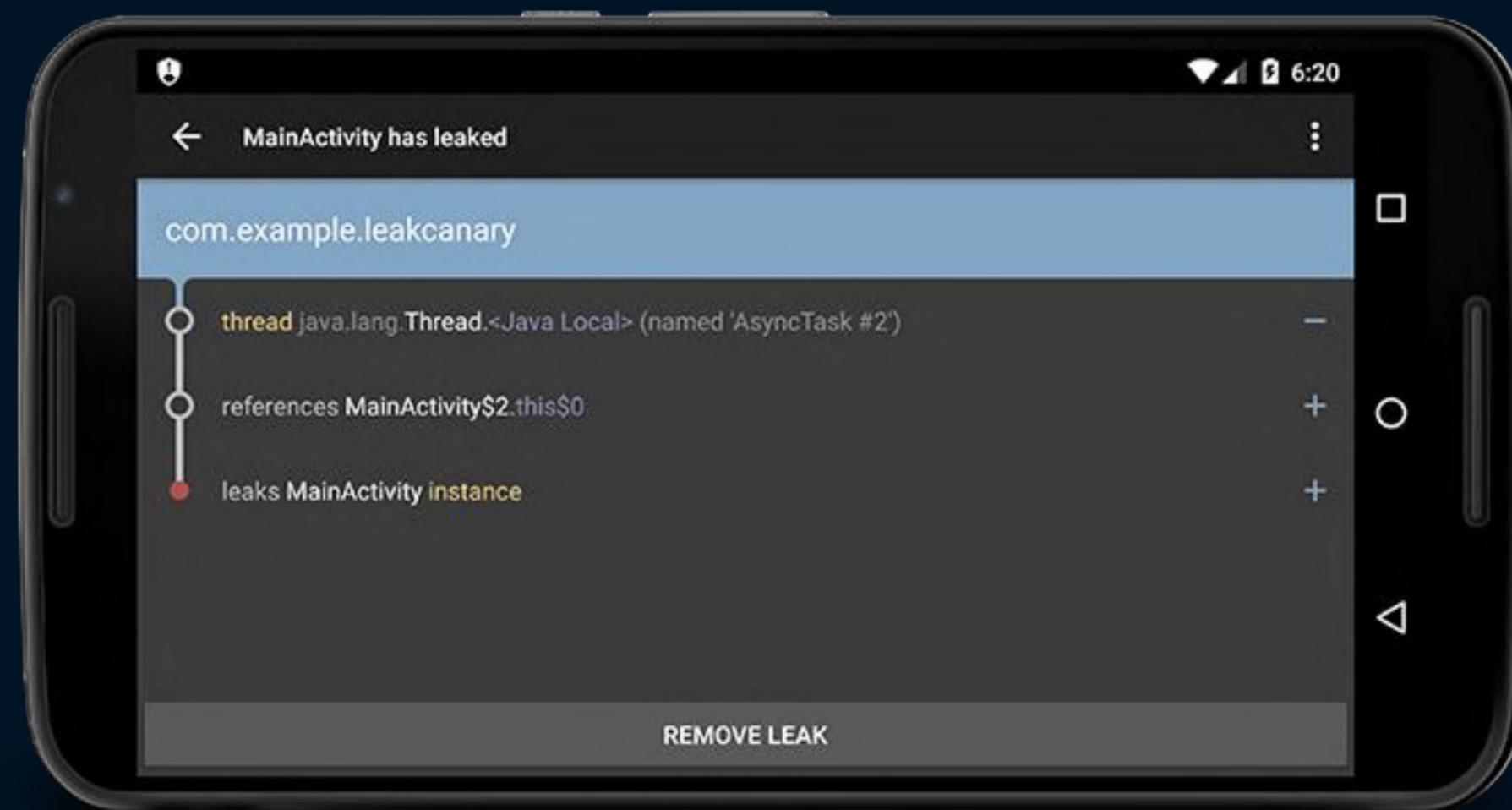
    @Override public void onDestroy() {
        super.onDestroy();
        RefWatcher refWatcher = ExampleApplication.getRefWatcher(getActivity());
        refWatcher.watch(this);
    }
}
```

Update your dependencies to the latest SNAPSHOT (see build.gradle):

```
dependencies {
    debugCompile 'com.squareup.leakcanary:leakcanary-android:1.5-SNAPSHOT'
    releaseCompile 'com.squareup.leakcanary:leakcanary-android-no-op:1.5-SNAPSHOT'
}
```

Add Sonatype's snapshots repository:

```
repositories {
    mavenCentral()
    maven {
        url 'https://oss.sonatype.org/content/repositories/snapshots/'
    }
}
```



# 目录

## CONTENT

1. 性能优化有多重要？
2. 应用性能状况
3. 性能优化流程、原则、指标和工具
4. 性能优化实战
5. 实际优化效果

## 04 性能优化实战-启动速度优化-三种启动模式

三种启动模式	
首次启动	耗时最长，fork进程+生成数据+dex编译为本地代码+应用初始化
冷启动	fork进程+应用初始化
热启动	应用恢复

应该以冷启动速度为优化指标，首次启动频次低，热启动耗时少，没多大意义。

# 04 性能优化实战-启动速度优化-冷启动平均耗时统计

[NimbleDroid](#)统计Google Play各类别APP冷启动平均耗时  
(Nexus 5 + Android 4.4)

类别	冷启动平均耗时 (秒)
在线音乐	3
通讯	2.5
购物	2.6
社交媒体	2.65
在线视频	3.5
设备优化	2.2
图像编辑	3.7



Google Play排名前100的非游戏类应用  
39个冷启动时间在2秒以内  
73个冷启动时间在3秒以内

# 04 性能优化实战-启动速度优化

启动速度优化		
启动分类	首次启动、冷启动、热启动	
现象	黑白屏、首帧出现慢	
原因分析	布局复杂、UI阻塞、过渡绘制	
测试工具	秒表、高速相机、命令行、Log(SDK≥4.4)	
检测工具	StrictMode、TraceView、Hugo、Hierarchy Viewer	
检测方法	使用StrictMode、Hugo、Hierarchy Viewer初步定位 使用TraceView精确定位	
优化方法	布局复杂	去掉Window背景、merge、ViewStub、space、CoordinatorLayout;
	过渡绘制	
	UI阻塞	AsyncTask、HandlerThread、IntentService、ThreadPool

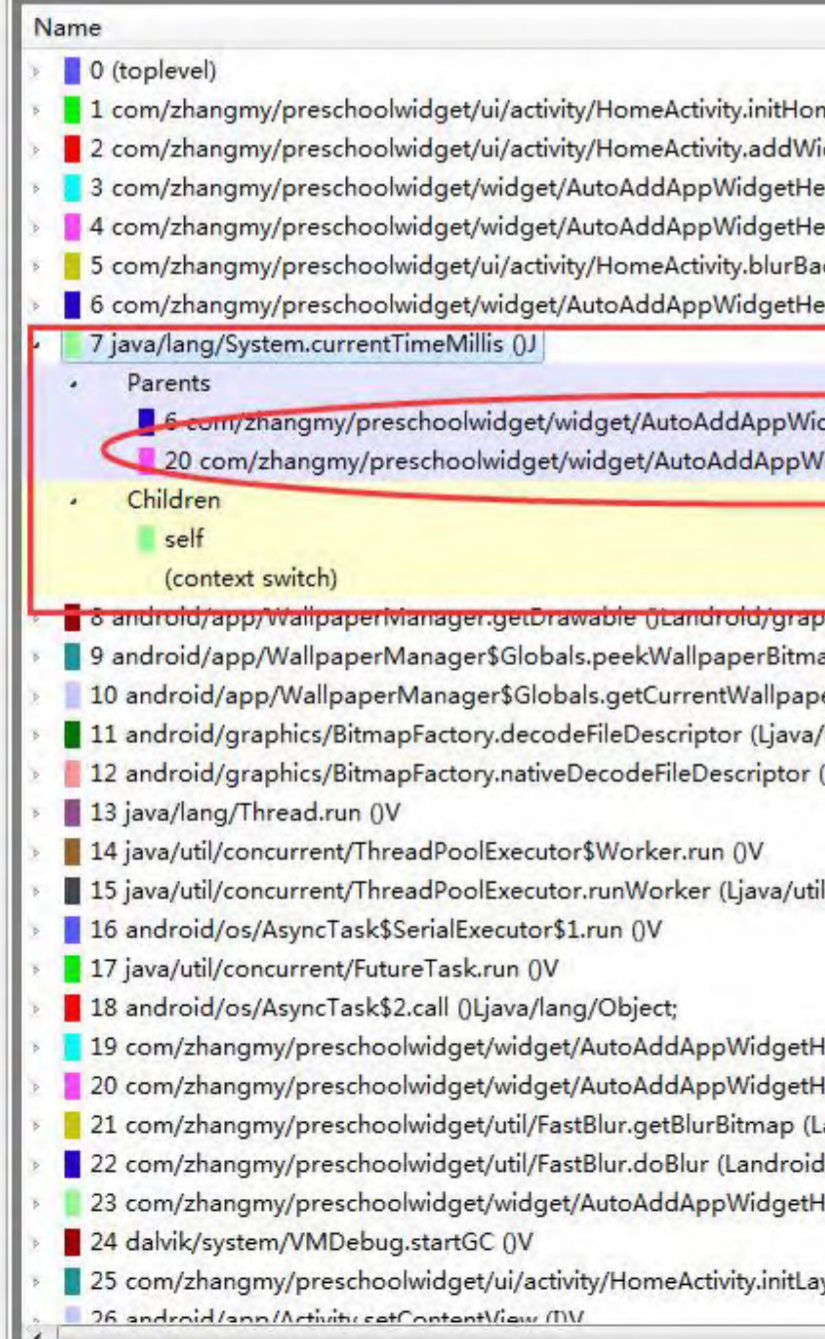
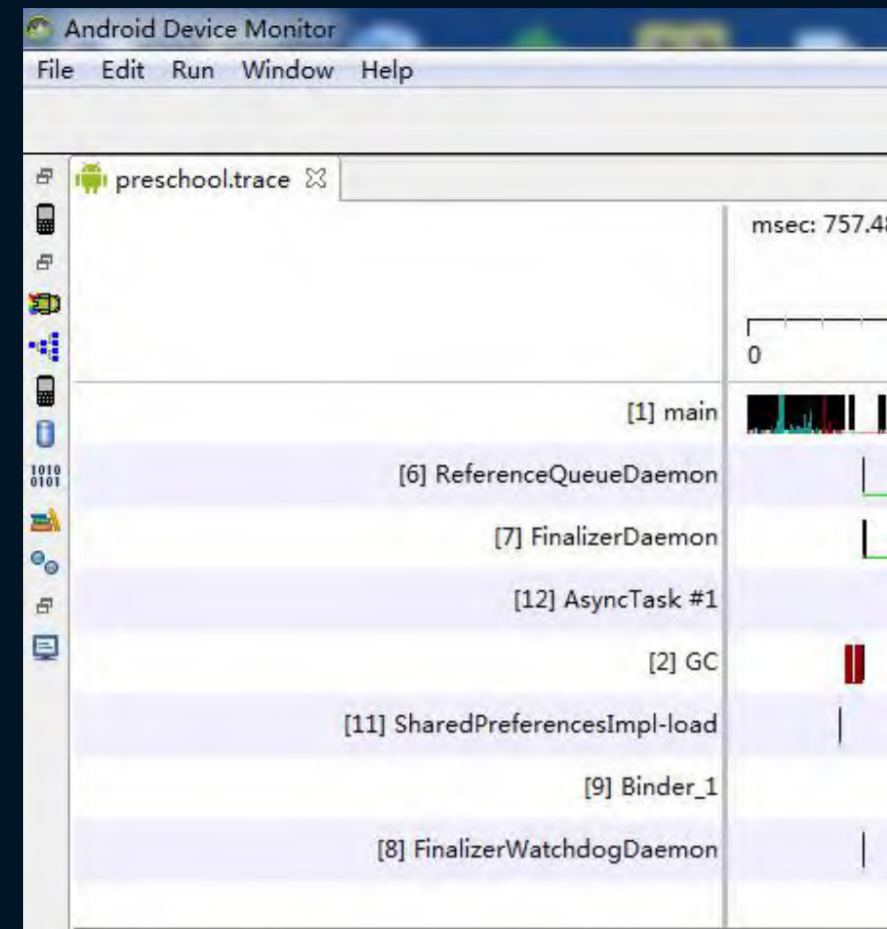
```
/system_process I/ActivityManager: [AppLaunch] Displayed Displayed com.eebk.bbkmiddlemarket/.activity.HomeActivity: +1s262ms
```

# 04 性能优化实战-启动速度优化-Hugo大致定位耗时位置

```
public class HomeActivity extends FragmentActivity implements AppFragment.OnFragmentInteractionListener{  
  
    // ...这里是实例化对象的操作  
  
    @DebugLog  
    @Override  
    protected void onCreate(Bundle savedInstanceState){  
        super.onCreate(savedInstanceState);  
        initHome();  
    }  
  
    @DebugLog  
    private void initHome(){  
        initLayout();  
        addWidget2Launcher();  
        blurBackground();  
        showContent();  
    }  
  
    @DebugLog  
    private void initLayout(){setContentView(R.layout.activity_home_layout);}  
  
    @DebugLog  
    private void addWidget2Launcher(){...}  
  
    @DebugLog  
    private void blurBackground(){...}  
  
    @DebugLog  
    private void showContent(){  
        findViews();  
        showTitle();  
        setViewPager();  
    }  
  
    @DebugLog  
    private void findViews(){...}  
  
    @DebugLog  
    private void setViewPager(){...}
```

```
09-19 09:47:07.912 13908-13908/? V/PreSchoolWidgetApplication: ← onCreate [0ms]  
09-19 09:47:08.012 13908-13908/? V/HomeActivity: ← initLayout [61ms]  
09-19 09:47:09.882 13908-13908/? V/HomeActivity: ← addWidget2Launcher [1868ms]  
09-19 09:47:11.492 13908-13908/? V/HomeActivity: ← blurBackground [1610ms]  
09-19 09:47:11.502 13908-13908/? V/HomeActivity: ← findViews [0ms]  
09-19 09:47:11.502 13908-13908/? V/HomeActivity: ← showTitle [1ms]  
09-19 09:47:11.512 13908-13908/? V/HomeActivity: ← setViewPager [5ms]  
09-19 09:47:11.512 13908-13908/? V/HomeActivity: ← showContent [14ms]  
09-19 09:47:11.512 13908-13908/? V/HomeActivity: ← initHome [3561ms]  
09-19 09:47:11.512 13908-13908/? V/HomeActivity: ← onCreate [3562ms]
```

# 04 性能优化实战-启动速度优化-TraceView精确定位

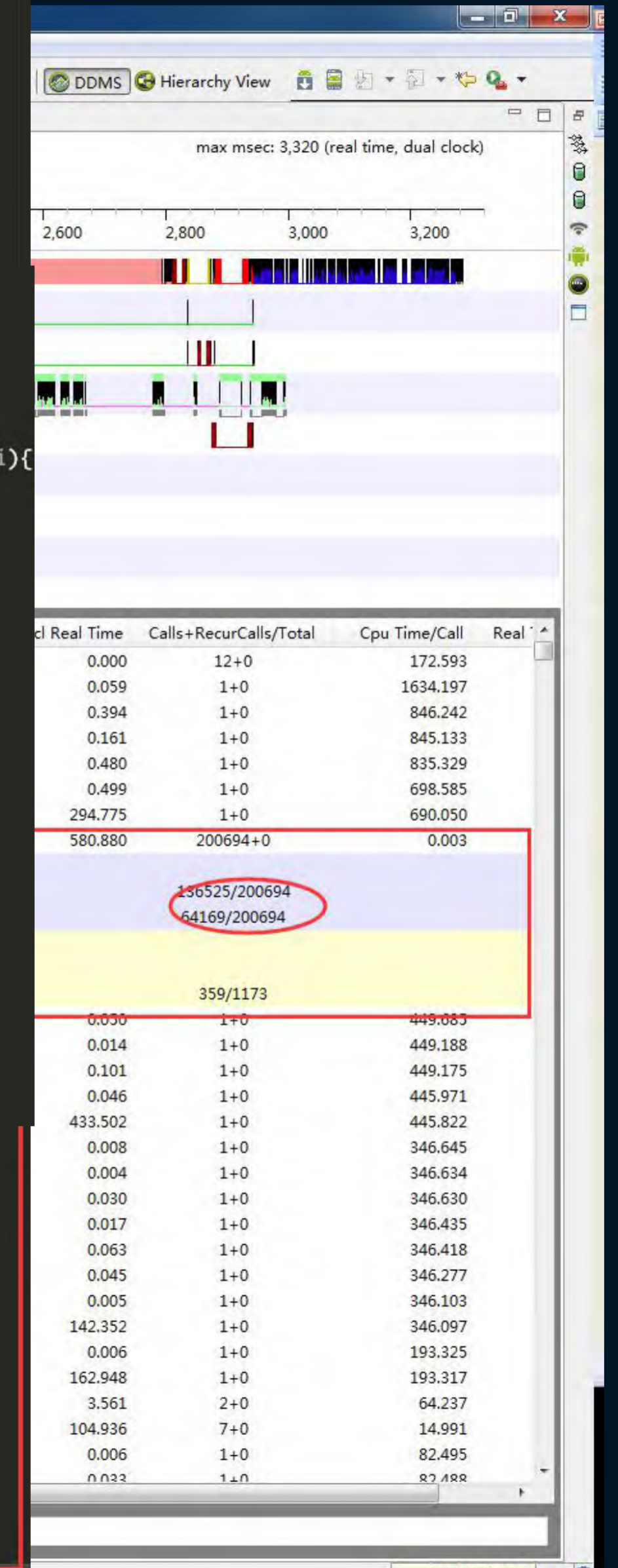


```
private static boolean isNewLauncherSupportAutoAdd(ContentResolver contentResolver){
    Cursor cursor = null;
    int pCount, jCount;
    int pNewCount, jNewCount;
    try{
        cursor = contentResolver.query(PRESCHOOL_CONTENT_URI, null, null, null, null);
        if(cursor == null){
            return false;
        }
    }
    static class CheckAddStateTask extends AsyncTask<Void, Void, Void>{
        private Context context;
        private ContentResolver resolver;
        private AppWidgetManager appWidgetManager;
        private Uri uri;

        public CheckAddStateTask(Context context, ContentResolver resolver, AppWidgetManager appWidgetManager, Uri uri){
            this.resolver = resolver;
            this.appWidgetManager = appWidgetManager;
            this.uri = uri;
            this.context = context;
        }

        @Override
        protected Void doInBackground(Void... params){
            long startTime = System.currentTimeMillis();
            while(true){
                long currentTime = System.currentTimeMillis();
                if(currentTime - startTime > 5000){
                    if(isAlreadyAddToDesktop(resolver, appWidgetManager, uri)){
                        setHasAutoAddAppWidget(context, true);
                    }else{
                        setHasAutoAddAppWidget(context, false);
                    }
                }
                return null;
            }
        }
    }

    cursor = contentResolver.query(JUNIOR_CONTENT_URI, null, null, null, null);
    if(cursor != null){
        jNewCount = cursor.getCount();
        if(jNewCount != jCount){
            return false;
        }
    }
    break;
}
```





# 04 性能优化实战-启动速度优化-改善后

```
private static boolean isNewLauncherSupportAutoAdd(ContentResolver contentResolver){
    Cursor cursor = null;
    int pCount, jCount;
    int pNewCount, jNewCount;
    try{
        cursor = contentResolver.query(PRESCHOOL_CONTENT_URI, null, null, null, null);
        if(cursor == null){
            return false;
        }else{
            if(cursor.getCount() == 0){
                return false;
            }
            pCount = cursor.getCount();
        }
        cursor = contentResolver.query(JUNIOR_CONTENT_URI, null, null, null, null);
        if(cursor == null){
            return false;
        }else{
            if(cursor.getCount() == 0){
                return false;
            }
            jCount = cursor.getCount();
        }
    }

    long currentTime = System.currentTimeMillis();
    while(true){
        if(System.currentTimeMillis() - currentTime > 1000){
            cursor = contentResolver.query(PRESCHOOL_CONTENT_URI, null, null, null, null);
            if(cursor != null){
                pNewCount = cursor.getCount();
                if(pNewCount != pCount){
                    return false;
                }
            }
            cursor = contentResolver.query(JUNIOR_CONTENT_URI, null, null, null, null);
            if(cursor != null){
                jNewCount = cursor.getCount();
                if(jNewCount != jCount){
                    return false;
                }
            }
        }
        break;
    }
}
```

```
private static boolean isNewLauncherSupportAutoAdd(ContentResolver contentResolver){
    Cursor cursor = null;
    int pCount, jCount;
    int pNewCount, jNewCount;
    try{
        cursor = contentResolver.query(PRESCHOOL_CONTENT_URI, null, null, null, null);
        if(cursor == null){
            return false;
        }else{
            if(cursor.getCount() == 0){
                return false;
            }
            pCount = cursor.getCount();
        }
        cursor = contentResolver.query(JUNIOR_CONTENT_URI, null, null, null, null);
        if(cursor == null){
            return false;
        }else{
            if(cursor.getCount() == 0){
                return false;
            }
            jCount = cursor.getCount();
        }
    }

    cursor = contentResolver.query(PRESCHOOL_CONTENT_URI, null, null, null, null);
    if(cursor != null){
        pNewCount = cursor.getCount();
        if(pNewCount != pCount){
            return false;
        }
    }
    cursor = contentResolver.query(JUNIOR_CONTENT_URI, null, null, null, null);
    if(cursor != null){
        jNewCount = cursor.getCount();
        if(jNewCount != jCount){
            return false;
        }
    }
}
```

由监听开机广播改为监听解锁的广播

# 04 性能优化实战-启动速度优化-改善后

```
static class CheckAddStateTask extends AsyncTask<Void, Void, Void>{
    private Context context;
    private ContentResolver resolver;
    private AppWidgetManager appWidgetManager;
    private Uri uri;

    public CheckAddStateTask(Context context, ContentResolver resolver, AppWidgetManager appWidgetManager, Uri uri){
        this.resolver = resolver;
        this.appWidgetManager = appWidgetManager;
        this.uri = uri;
        this.context = context;
    }

    @Override
    protected Void doInBackground(Void... params){
        long startTime = System.currentTimeMillis();
        while(true){
            long currentTime = System.currentTimeMillis();
            if(currentTime - startTime > 5000){
                if(isAlreadyAddToDesktop(resolver, appWidgetManager, uri)){
                    setHasAutoAddAppWidget(context, true);
                }else{
                    setHasAutoAddAppWidget(context, false);
                }
                return null;
            }
        }
    }
}
```

```
static class CheckAddStateTask extends AsyncTask<Void, Void, Void>{
    private Context context;
    private ContentResolver resolver;
    private AppWidgetManager appWidgetManager;
    private Uri uri;

    public CheckAddStateTask(Context context, ContentResolver resolver, AppWidgetManager appWidgetManager, Uri uri){
        this.resolver = resolver;
        this.appWidgetManager = appWidgetManager;
        this.uri = uri;
        this.context = context;
    }

    @Override
    protected Void doInBackground(Void... params){
        if(isAlreadyAddToDesktop(resolver, appWidgetManager, uri)){
            setHasAutoAddAppWidget(context, true);
        }else{
            setHasAutoAddAppWidget(context, false);
        }
        return null;
    }
}
```

延时的目的是为了解决刚开机时Launcher没有加载完成导致查询数据失败的问题  
由监听开机广播改为监听解锁的广播。

## 04 性能优化实战-启动速度优化-改善后

```
09-19 09:47:07.912 13908-13908/? V/PreSchoolWidgetApplication: ← onCreate [0ms]
09-19 09:47:08.012 13908-13908/? V/HomeActivity: ← initLayout [61ms]
09-19 09:47:09.882 13908-13908/? V/HomeActivity: ← addWidget2Launcher [1868ms]
09-19 09:47:11.492 13908-13908/? V/HomeActivity: ← blurBackground [1610ms]
09-19 09:47:11.502 13908-13908/? V/HomeActivity: ← findViews [0ms]
09-19 09:47:11.502 13908-13908/? V/HomeActivity: ← showTitle [1ms]
09-19 09:47:11.512 13908-13908/? V/HomeActivity: ← setViewPager [5ms]
09-19 09:47:11.512 13908-13908/? V/HomeActivity: ← showContent [14ms]
09-19 09:47:11.512 13908-13908/? V/HomeActivity: ← initHome [3561ms]
09-19 09:47:11.512 13908-13908/? V/HomeActivity: ← onCreate [3562ms]
```

```
09-19 16:11:31.272 459-459/? V/HomeActivity: ← initLayout [111ms]
09-19 16:11:31.622 459-459/? V/HomeActivity: ← addWidget2Launcher [352ms]
09-19 16:11:32.632 459-459/? V/HomeActivity: ← blurBackground [1011ms]
09-19 16:11:32.642 459-459/? V/HomeActivity: ← findViews [0ms] 时间减少了近2秒
09-19 16:11:32.642 459-459/? V/HomeActivity: ← setViewPager [3ms]
09-19 16:11:32.642 459-459/? V/HomeActivity: ← showContent [8ms]
09-19 16:11:32.642 459-459/? V/HomeActivity: ← initHome [1488ms]
09-19 16:11:32.642 459-459/? V/HomeActivity: ← onCreate [1490ms]
```

# 04 性能优化实战-流畅度优化

## 流畅度优化

现象	应用卡顿	
原因分析	内存抖动、过渡绘制、UI阻塞、整机剩余内存不足	
测试工具	Android Studio、开发者选项	
检测工具	StrictMode、TraceView、Hugo、Android Studio、开发者选项	
检测方法	使用StrictMode、Hugo、Android Studio、开发者选项初步定位 使用TraceView精确定位	
优化方法	内存抖动	减少局部变量的申请，特别是在频繁调用方法内；
	过渡绘制	去掉Window背景、merge、ViewStub、space、CoordinatorLayout；
	UI阻塞	AsyncTask、HandlerThread、IntentService、ThreadPool
	整机内存	内存泄露优化（非静态内部类、静态代码检查、Context、系统SDK泄露）

# 04 性能优化实战-流畅度优化

## 内存优化

现象	整机慢	
原因分析	内存占用大、内存泄漏、内存抖动	
测试工具	Android Studio、Leakcanary	
检测工具	Android Studio、Leakcanary	
检测方法	使用adb shell dumpsys meminfo初步定位问题 使用Android Memory Monitor、Leakcanary精确定位问题	
优化方法	内存占大	缓存合理，图片放置位置合理。
	内存泄露	内存泄露优化（非静态内部类、静态代码检查、Context、系统SDK泄露）
	内存抖动	减少局部变量的申请，特别是在频繁调用方法内；



# 04 性能优化实战-功耗优化

## 功耗优化

现象	发烫、掉电快	
原因分析	硬件持续高负荷工作 如频繁的网络访问、频繁的界面刷新、长时间不待机、后台持续工作	
测试工具	万用表、专业的电量测试工具	
检测工具	Android Studio、开发者选项	
检测方法	使用Android Monitor初步定位网络、C/GPU的问题 使用开发者选项精确定位频繁刷新的问题	
优化方法	频繁的网络访问	网络请求合并，改善网络请求策略
	频繁的界面刷新	改为触发时响应，不要一进界面就持续刷新界面
	长时间不待机	合理使用WakeLock
	后台持续工作	系统优化

# 目录

## CONTENT

1. 性能优化有多重要？
2. 应用性能状况
3. 性能优化流程、原则、指标和工具
4. 性能优化实战
5. 实际优化效果

## 05 实际优化效果

- 超过80%的应用冷启动速度基本都控制在了1.5S以内
- 解决图文混排控件界面因为图片太多导致滑动卡顿的问题
- 对大部分模块完成静态代码分析，解决了明显的性能问题（Handler内部类、IO操作导致的内存泄露）
- 解决两个APP因为适配不合理（图片存放的drawable文件夹不对）导致耗内存的问题
- 超过80%的应用基本做到无内存泄漏（除系统导致的内存泄漏问题外）
- 解决了两个系统导致的内存泄漏问题（WebView、InputMethodManager）
- 超过80%的应用过渡绘制层数控制在3X以内（无红色）
- 超过95%的应用安装包大小控制在10M以内



我讲完了，谢谢大家

**THANKS!**

