M1000应用性能优化经验分享

目录

说明

100%的应用存在性能问题！

说明

主要性能指标：

指标	现象	原因	检测工具
内存	卡 OOM 整机慢	内存抖动 代码不合理 内存泄露	Android Studio→Android Monitor Android Studio→Analyze→Inspect Code… Leakcanary MAT
流畅度	卡 慢(页面切换慢、启动慢) 黑白屏 ANR	内存抖动 UI阻塞 过渡绘制 跨进程	Android Studio→Android Monitor 开发者选项→过渡绘制 开发者选项→呈现模式分析 开发者选项→启用严格模式StrictMode
功耗	发热 掉电快	频繁的网络访问 CPU高负荷工作 长时间不待机 频繁的界面刷新	Android Studio→Android Monitor
体积	耗流量	图片资源 无用资源 预置数据没压缩	Android Studio→Build→Analyze APK…

说明

工作成效

- 对大部分模块（拿得到代码的）完成静态代码分析，解决了明显的性能问题；
- 解决两个APP耗内存大的问题：问题反馈、随心壁纸；
- 解决九科同步内容界面严重卡顿的问题；
- 改善名师辅导班、应用商城、好题库、问题反馈、随心壁纸等模块的内存泄露问题，做到无内存泄露；
- 发现两个系统SDK导致的内存泄露：WebView、InputMethodManager；
- 除集成在系统中的几个APK外，其它APK的过渡绘制层数基本都控制在3X以内（极少有红色）；
- 33个APK；APK>50M 0个；APK>40M 1个；APK>30M 1个；APK>20M 5个；APK>10M 10个；

说明

普遍存在的问题：

1、非静态内部类导致的内存泄露（Handler、Observer）；

2、四大组件的Context与Application Contxt的不合理使用；

3、几乎所有模块都存在过渡绘制，背景设置了两遍：window和activity都设置了背景；

4、IO操作完成后没有关闭文件，TypedArray使用完成后没有recycle，Cursor使用完成后没有close；

5、String、StringBuilder、StringBuffer；SpareArray、HashMap;for loop;系统方法、自己实现的代码块（arraycopy）；基础类型、类（boolean、Boolean）；

6、在系统回调中创建新的实例（eg: onDraw、getView）；

7、系统SDK导致的泄露：WebView、InputMethodManager；

说明

用到的工具：

- Android Studio;
- 开发者选项（需要安装原生系统设置APK）;
- LeakCanary;
- MAT;
- DDMS;
- tinypng、iSparta、7zip

说明

关于GC：

- GC是在非UI线程中进行的，但GC的过程会暂停所有线程，所以频繁的GC会导致线程阻塞，存在卡顿的问题；
- GC的目的可以理解为释放内存；
- 一个内存泄露问题会导致与之相关的类都GC不掉，即一个内存泄露问题可能导致上M的内存无法及时回收；
- 关于JVM内存管理参见：Android GC 那点事

说明

Android开发者选项GPU呈现模式：

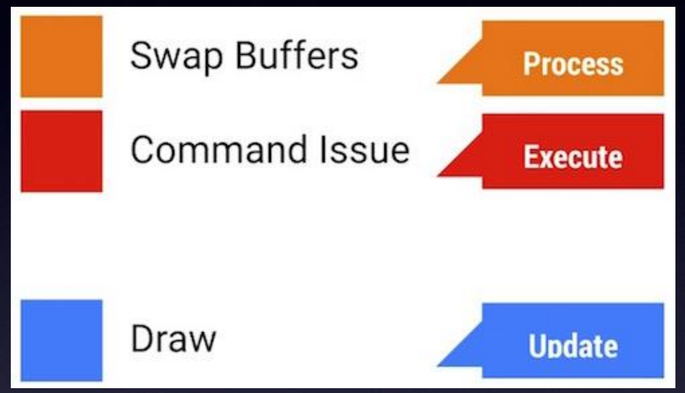


图1：API>=4.1

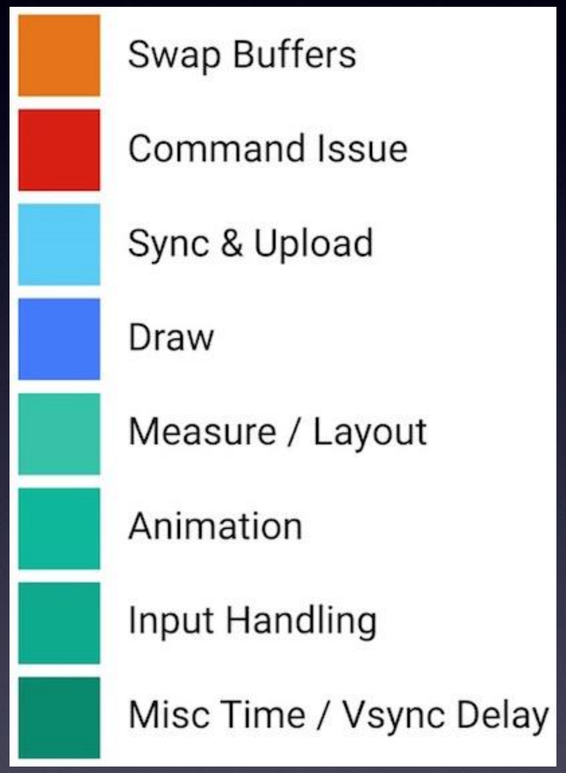


图1：API>=6.0

Swap Buffers：表示处理任务的时间，也可以说是CPU等待GPU完成任务的时间，线条越高，表示GPU做的事情越多；

Command Issue：表示执行任务的时间，这部分主要是Android进行2D渲染显示列表的时间，为了将内容绘制到屏幕上，Android需要使用Open GL ES的API接口来绘制显示列表，红色线条越高表示需要绘制的视图更多；

Sync & Upload：表示的是准备当前界面上有待绘制的图片所耗费的时间，为了减少该段区域的执行时间，我们可以减少屏幕上的图片数量或者是缩小图片的大小；

Draw：表示测量和绘制视图列表所需要的时间，蓝色线条越高表示每一帧需要更新很多视图，或者View的onDraw方法中做了耗时操作；

Measure/Layout：表示布局的onMeasure与onLayout所花费的时间，一旦时间过长，就需要仔细检查自己的布局是不是存在严重的性能问题；

Animation：表示计算执行动画所需要花费的时间，包含的动画有ObjectAnimator，ViewPropertyAnimator，Transition等等。一旦这里的执行时间过长，就需要检查是不是使用了非官方的动画工具或者是检查动画执行的过程中是不是触发了读写操作等等；

Input Handling：表示系统处理输入事件所耗费的时间，粗略等于对事件处理方法所执行的时间。一旦执行时间过长，意味着在处理用户的输入事件的地方执行了复杂的操作；

Misc Time/Vsync Delay：表示在主线程执行了太多的任务，导致UI渲染跟不上vSync的信号而出现掉帧的情况；出现该线条的时候，可以在Log中看到这样的日志：

静态代码分析

目的：

- 发现代码层面的性能问题，非运行时；

工具：

- Android Studio;

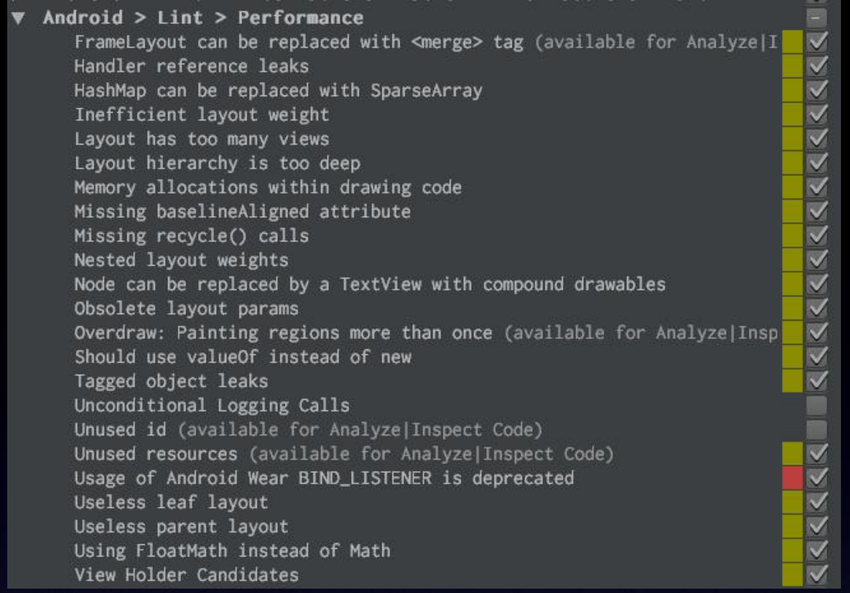
步骤

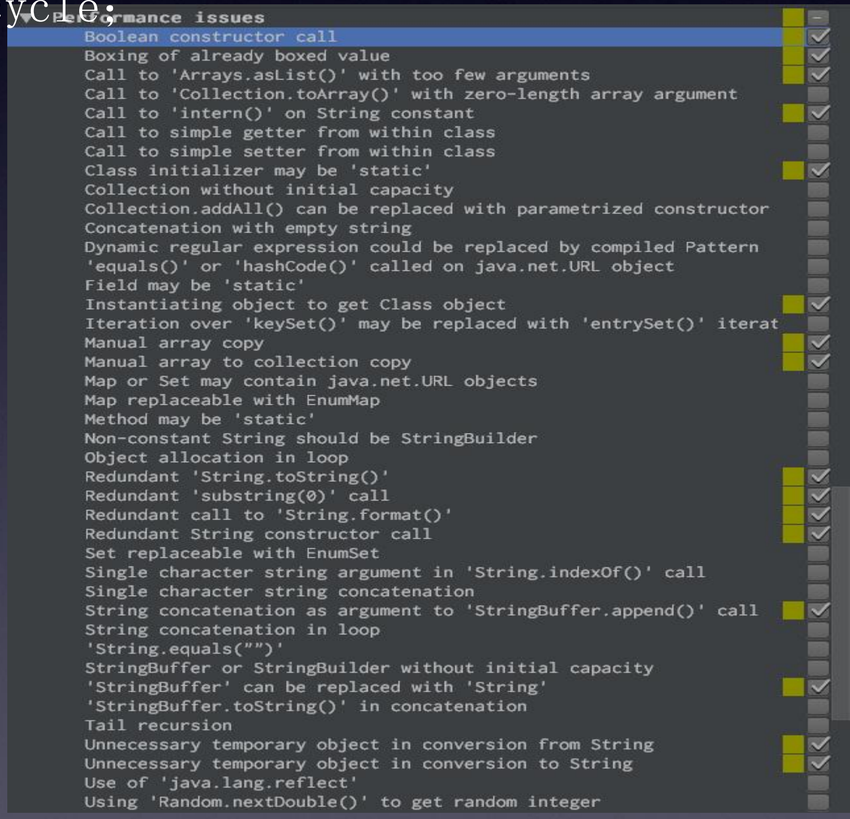
- Analyze→Inspect Code…

静态代码分析

能够发现的问题：

- Handler内部类导致的Leak；
- IO操作完成后没有关闭文件；Cursor没有关闭；TypedArray没有recycle；
- 布局的性能问题（过渡绘制、更高效的布局建议、useless标签）；
- Hashmap替换成SpaseArray，SparseArray vs HashMap；
- Unused resources；
- 系统回调中创建新的实例；
- String、StringBuilder、StringBuffer不合理使用的问题；
- More。。。





内存优化

现象：

- 耗内存大；
- 退出页面、完全退出程序后GC不掉占用的内存；
- 进入、退出页面和程序前后，手动触发GC，adb shell dumpsys meminfo packagename，Activities和Views的数量有差异；
- 卡顿，LOG中GC_FOR_ALLOC、GC_CONCURRENT等打印消息；
- 程序在后台时很容易被回收；

工具

- LeakCanary；
- MAT；

内存优化

步骤：

- 解决AS静态代码分析结果中的内存泄露问题；
- 解决常见的内存泄露问题；
- 解决耗内存大的问题；
- 先定位问题，再用MAT分析；

内存优化

步骤： 解决AS静态代码分析结果中的内存泄露问题；

- IO操作完成后没有关闭文件；TypedArray没有release；Cursor没有关闭；
- Handler内部类导致的内存泄露问题；

```
class ButtonHandler extends Handler {
    private static final int MSG_DISMISS_DIALOG = 1;

    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {

            case DialogInterface.BUTTON_POSITIVE:
            case DialogInterface.BUTTON_NEGATIVE:
            case DialogInterface.BUTTON_NEUTRAL:
                ((DialogInterface.OnClickListener) msg.obj).onClick(AlertController.this, msg.what);
                break;

            case MSG_DISMISS_DIALOG:
                ((DialogInterface) msg.obj).dismiss();
        }
    }
}
```

```
private static final class ButtonHandler extends Handler {
    private static final int MSG_DISMISS_DIALOG = 1;
    private WeakReference<DialogInterface> mDialog;

    public ButtonHandler(DialogInterface dialog) { mDialog = new WeakReference<>(dialog); }

    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case DialogInterface.BUTTON_POSITIVE:
            case DialogInterface.BUTTON_NEGATIVE:
            case DialogInterface.BUTTON_NEUTRAL:
                ((DialogInterface.OnClickListener) msg.obj).onClick(mDialog.get(), msg.what);
                break;
            case MSG_DISMISS_DIALOG:
                ((DialogInterface) msg.obj).dismiss();
        }
    }
}
```

内存优化

步骤： 解决常见的内存泄露问题；

- 四大组件的Context和Application Context

	Application	Activity	Service	ContentProvider	BroadcastReceiver
Show a Dialog	NO	YES	NO	NO	NO
Start an Activity	NO[1]	YES	NO[1]	NO[1]	NO[1]
Layout Inflation	NO[2]	YES	NO[2]	NO[2]	NO[2]
Start a Service	YES	YES	YES	YES	YES
Bind to a Service	YES	YES	YES	YES	NO
Send a Broadcast	YES	YES	YES	YES	YES
Register BroadcastReceiver	YES	YES	YES	YES	NO[3]
Load Resource Values	YES	YES	YES	YES	YES

带数字的NO：

1、从能力上来讲，是可以使用Application Context的，但得分情况；

2、1：启动Activity在这些类中是可以的，但是需要创建一个新的task，一般情况不推荐；

3、在这些类中去layout inflate是合法的，但是会使用系统默认的主题样式，如果你自定义了某些样式可能不会被使用；

4、在Receiver为null时允许，在4.2或以上的版本中，用于获取黏性广播的当前值（可以无视）；

内存优化

步骤： 解决耗内存大的问题；

- **定义**：耗内存大？没有确切的数字，大得离谱就叫大，M1000一张背景图占用近8M；如果一启动进程，就占用超过50M就可以定义为大；

- **观察**：Android Studio→Android Monitor→Monitors；

- **分析**：

1、最占用内存的是图片，先对图片资源进行分析；→相同图片，放在不同的drawable文件夹下，在同一个设备中占用的内存不一样（涉及到图片缩放）；Android drawable微技巧，你所不知道的drawable的那些细节

2、排除图片问题后，再对代码进行分析，是否有大量的数据、对象放在内存中。

- **建议**：

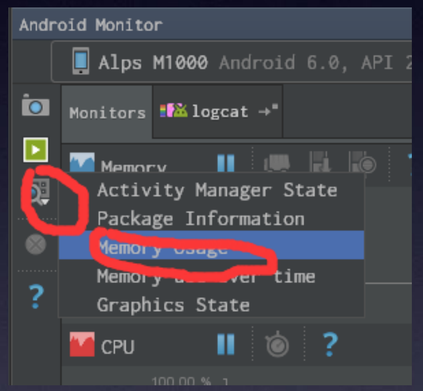
1、在Activity的onTrimMemory方法中做好释放内存的处理；合理管理内存

内存优化

步骤： 先定位问题，再用MAT分析；

- 定位问题：

1、命令：退出页面/程序，手动触发GC后，执行adb shell dumpsys meminfo packagename。




```
Objects
                 Views:      428        ViewRootImpl:        1
           AppContexts:        4          Activities:        1
                Assets:        4        AssetManagers:       2
          Local Binders:       29        Proxy Binders:      23
          Parcel memory:        6         Parcel count:      26
       Death Recipients:        0      OpenSSL Sockets:       0
```

2、工具：LeakCanary

```
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);
  }
}
```

```
public abstract class BaseFragment extends Fragment {

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

内存优化

步骤： 先定位问题，再用MAT分析；

- MAT分析：

1、MAT只能发现问题，不能定位问题；

2、工具：Android Studio→Android Monitor→Monitors→Memory→Dump Java Heap；MAT

3、步骤：

（1）完全退出程序，手动触发GC；

（2）通过Dump Java Heap取内存快照；

（3）通过sdk/platform-tools中的hprof-conv.exe将hprof文件转成MAT能够分析的hprof文件；

（4）通过MAT打开hprof文件→Calculate Precise Retained Size→Group by package→右键感兴趣的类→Merge Shortest Paths to GCRoots→exclude all plantom/weak/sof etc. references。

流畅度优化

现象、原因、解决方案：UI线程干活太多

- 卡顿；→GC（GC是在UI线程工作）、刷新不过来；→减少GC、一帧内少干活、刷新区域尽量小；
- 慢；→UI线程干活太多、环境限制（网络慢、存储设备访问效率低等）；→异步、缓存、按需读取；
- 黑屏、白屏；→加载布局耗时、UI线程干活多、操作频繁→异步、减少布局复杂度；
- ANR→UI阻塞

流畅度优化

工具：

- DDMS→TraceView；
- 开发者选项→过渡绘制、GPU呈现模式分析、ANR；
- 其它：StrictMode、hugo、BlockCanary

功耗优化

现象、原因、解决方案：

- 发热、掉电快→CPU、GPU持续工作（界面持续刷新、频繁的IO操作、网络访问等）、不必要的后台服务→只在需要刷新的时间、区域进行刷新操作；合理的缓存；集中网络访问，不要持续、不间断地访问网络；优先使用IntentService，确实需要才考虑Service；

工具：

- Android Studio→Android Monitor→Monitors→NetWork；
- 开发者选项→GPU呈现模式分析；

APK体积优化

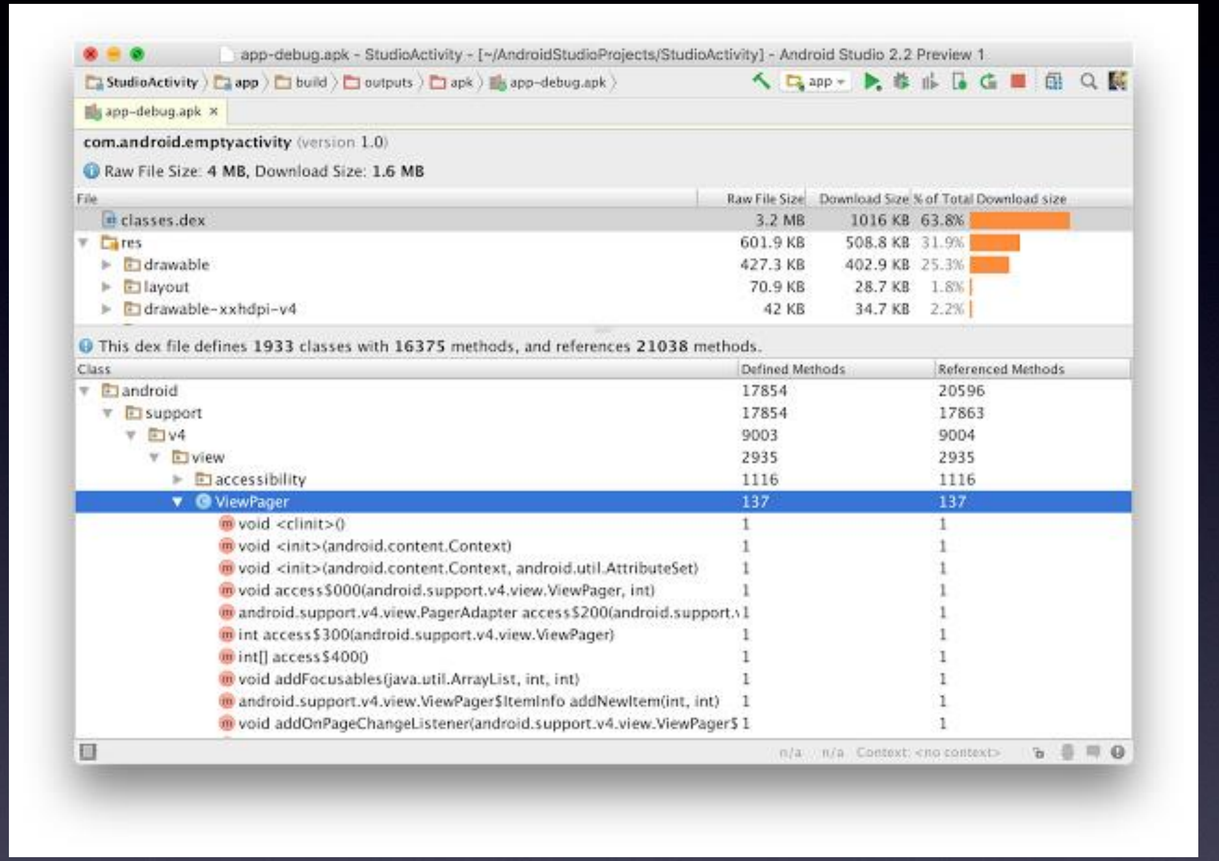
现象、原因、解决方案：

- APK文件大，会导致耗费更多地烧录时间、下载时间长、耗流量；
- 存在冗余的资源文件；
- 图片没有压缩；
- 代码和资源文件没有混淆；
- 集成到APK文件中的预置数据没有打包成压缩包；
- 音频资源太大，过于追求质量；

APK体积优化

工具：

- Android Studio→Inspect Code…;
- Android Studio→Build→Analyze APK…;
- www.tinypng.com;
- 图片压缩和格式转换工具iSparta ;
- 7zip，AndroidUn7zip ;
- Opus音频格式;



总结

资源推荐

- Android Performance Patterns（梯子：lantern）
- 15个必知的Android开发者选项
- Android客户端性能优化（魅族资深工程师毫无保留奉献）
- Speed up your app
- awesome-android-performance
- WebP 探寻之路

总结

忠告

- 很多性能问题都是可感知的，但优化过程中一定要看数据说话；
- 改善完成后一定要验证，避免因为改善性能问题导致程序不稳定；
- 如果优化过程中需要大改程序，建议以稳定为主，先保证可用，后保证好用；
- 性能优化是一个持续的过程；

我讲完了，鼓掌吧！