APP深度性能测试&性能提升实践

YunOS移动测试平台（MQC）

尚进

APP遇到的挑战

- 激烈竞争的市场
- 行为挑剔的用户
- 良莠不齐的设备

- 频繁的产品更新
- 复杂的代码框架
- 苛刻的性能指标

怎么做性能测试

线下

指定机型测试性能数据

按照业务测试固定的性能测试case，获取性能数据

线上

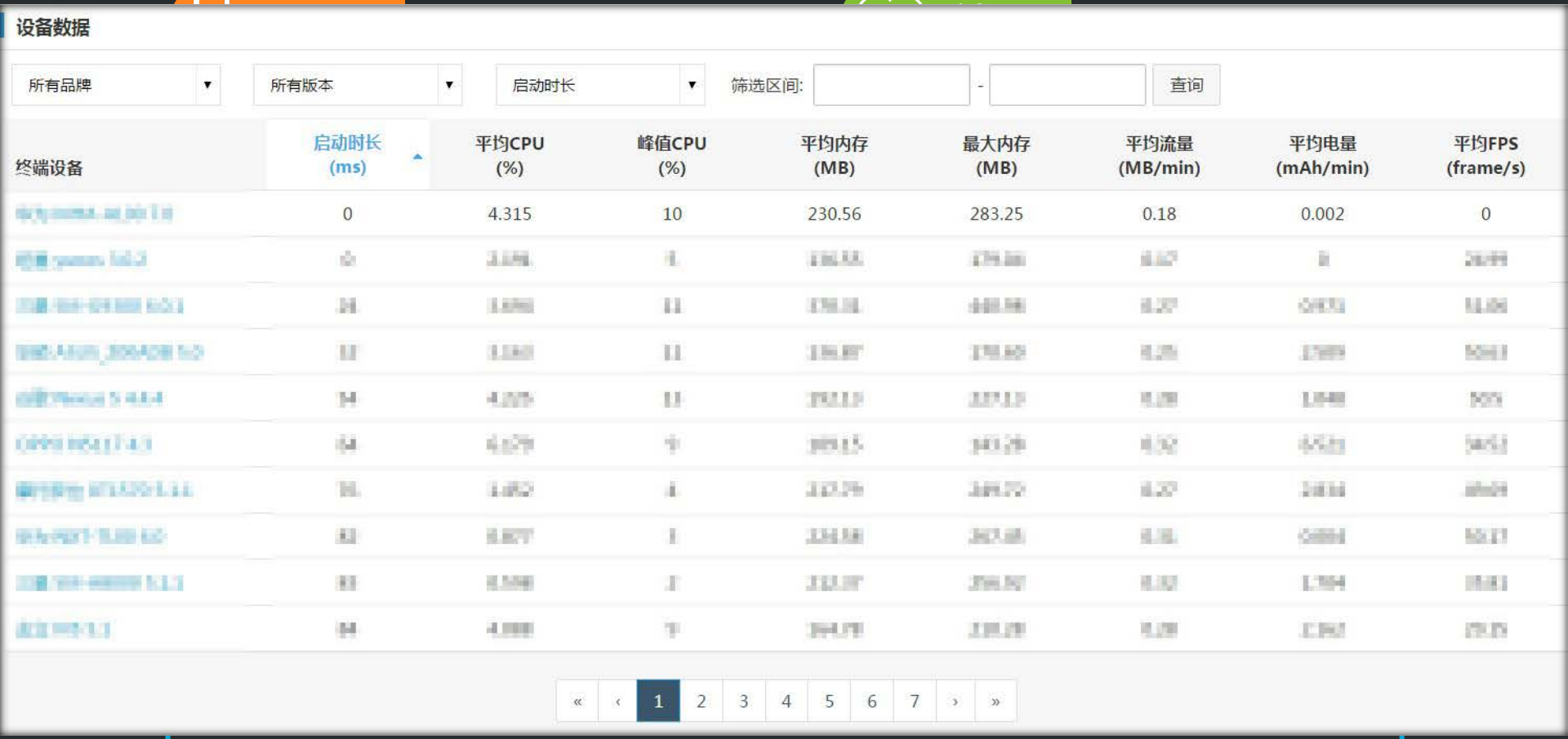
在线监控

监控APP运行性能基础数据，异常上报；
灰度开关

常用指标

启动时长、CPU占用、内存占用、
流量消耗、电量消耗、FPS、
页面渲染、GC次数、
……

怎么做性能测试


设备数据
所有品牌
所有版本
启动时长
筛选区间:
-
查询
终端设备
启动时长 (ms)
平均CPU (%)
峰值CPU (%)
平均内存 (MB)
最大内存 (MB)
平均流量 (MB/min)
平均电量 (mAh/min)
平均FPS (frame/s)
0
4.315
10
230.56
283.25
0.18
0.002
0
« ‹ 1 2 3 4 5 6 7 › »

MQC的深度性能测试

内存泄露&内存溢出

异常Activity引用

OOM日志检测、现场快照

内存趋势&GC监控

内存变化曲线

GC日志检测、分类

UI流畅度&卡顿监测

界面切换过程

掉帧记录和流畅度分析

GPU过度绘制

页面渲染层次

启动分析

冷热启动时间

启动分阶段分析

严苛模式StrictMode

主线程IO、Closeable ...

代码质量

内存泄露

GC ROOT

static variables
unfinished threads
instance of Application
Unclosed resources
…

Examples

```
public class TestActivity extends Activity {
    static Set<Object> set = new HashSet<Object>();
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        set.add(new Object());
    }
}
```

内存泄露

GC ROOT

static variables
unfinished threads
instance of Application
Unclosed resources
…

Examples

```
public class TestActivity extends Activity {
```

```
public class TestContextHelper {
    static TestContextHelper instance;
    private Context context;
    private TestContextHelper(Context context) {
        this.context = context;
    }
    public static TestContextHelper getInstance(Context context){
        if(instance != null){
            instance = new TestContextHelper(context);
        }
        return instance;
    }
}
```

内存泄露

GC ROOT

static variables
unfinished threads
instance of Application
Unclosed resources
…

Examples

```
public class TestActivity extends Activity {
```

```
public class TestContextHelper {
```

```
public class TestActivity extends Activity{
    Handler handler;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        handler = new Handler();
        //延时5分钟发送一个Message
        handler.sendMessageDelayed(new Message(), 300000);
    }
    class MyHandler extends Handler{
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
        }
    }
}
```

内存泄露

GC ROOT

static variables
unfinished threads
instance of Application
Unclosed resources
...

Examples

```
public class TestActivity extends Activity {
```

```
public class TestContextHelper {
```

```
public class TestActivity extends Activity{
    Handler handler;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
}
```

内存泄漏 | ⓘ发现内存泄漏 | 收起详情

机型	泄漏内存	泄漏类型	泄漏实例类型	操作
SAMSUNG GT-I9508-PERF 4.4 等1款	0.03 MB	Activity泄漏	com.stephen.leaks.MainActivity	查看引用链 终端详情
SAMSUNG GT-I9508-PERF 4.4 等1款	0.14 MB	Activity泄漏	com.stephen.leaks.OtherActivity	查看引用链 终端详情

```
* GC ROOT static com.stephen.leaks.ActivityManager.sInstance
* references com.stephen.leaks.ActivityManager.mActivities
* references java.util.ArrayList.array
* references array java.lang.Object[].[4]
* leaks com.stephen.leaks.OtherActivity instance
```

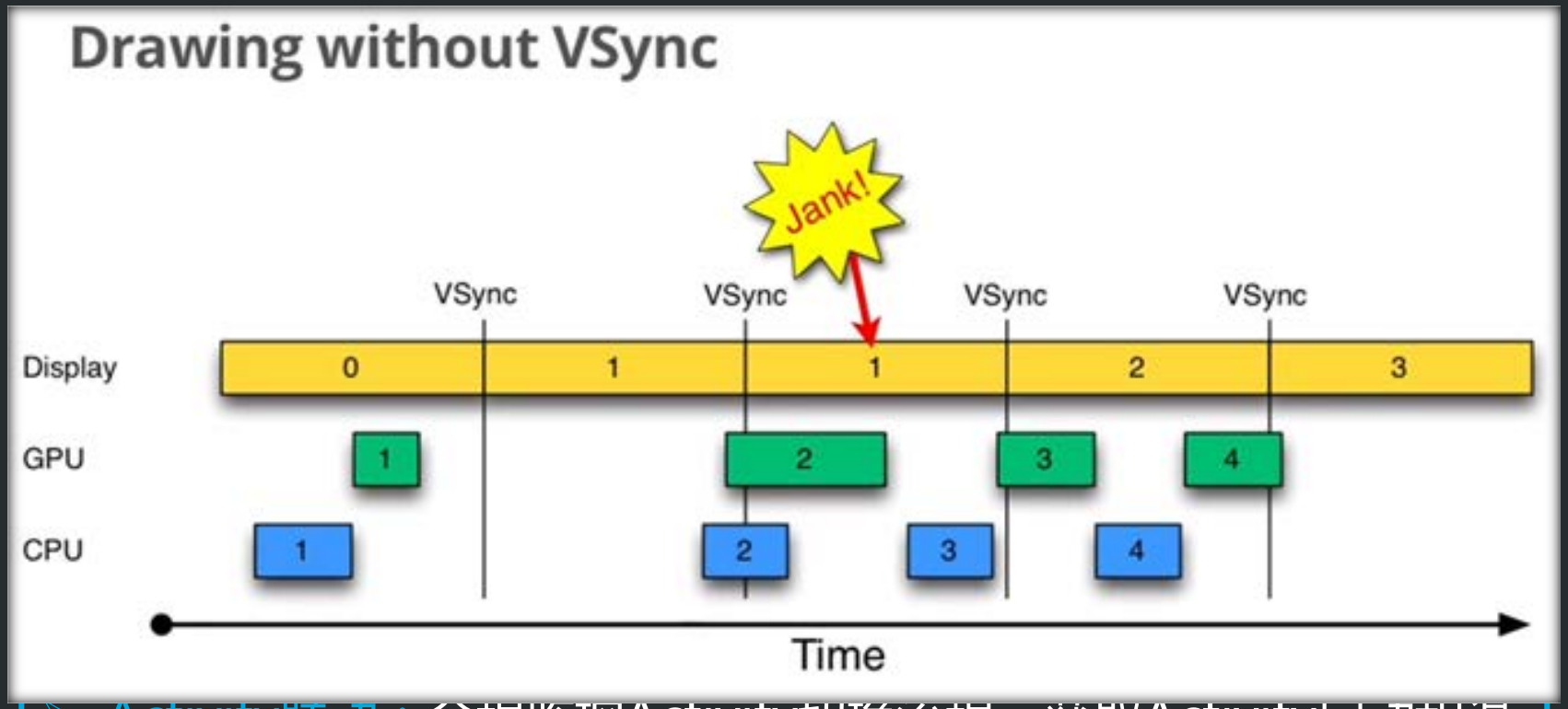
内存溢出

- Java虚拟机的内存限制
- 内存泄漏
- 内存消耗大户
  - Bitmap
  - …

DDMS, MAT…

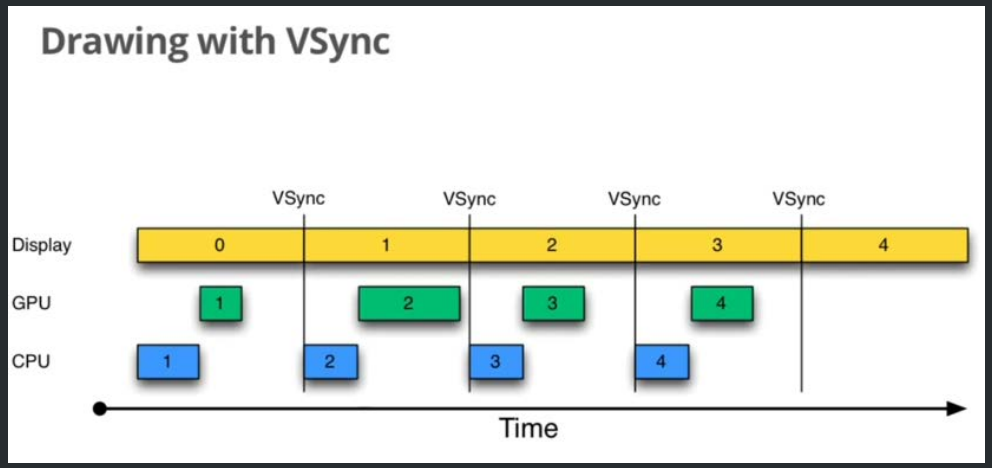
- ✓ 巧妙地使用WeakReference
- ✓ 及时销毁Bitmap
- ✓ 记得关闭Cursor

UI流畅度&卡顿



- Activity联动：全程监控Activity切换流程，获取Activity启动和退出耗时
- 卡顿联动：监控Message的处理时长，记录长时间阻塞UI线程的操作

UI流畅度&卡顿



丢帧率算法

通过Choreographer.FrameCallback的onFrame，获取当前帧率的时间，得出skipped Frame，推算出流畅度。

MORE

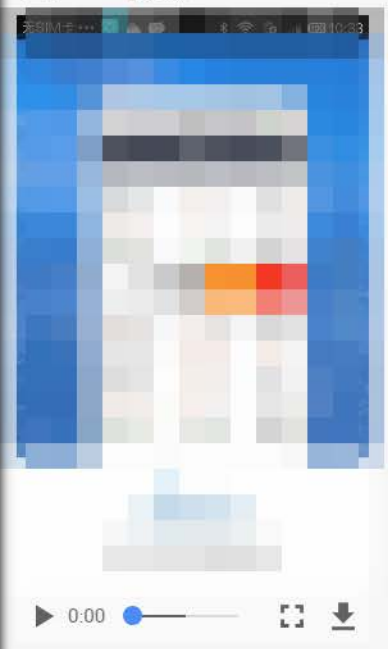
- 视频联动：录制APP运行过程视频，精确定位异常界面变化及前后操作
- Activity联动：全程监控Activity切换流程，获取Activity启动和退出耗时
- 卡顿联动：监控Message的处理时长，记录长时间阻塞UI线程的操作

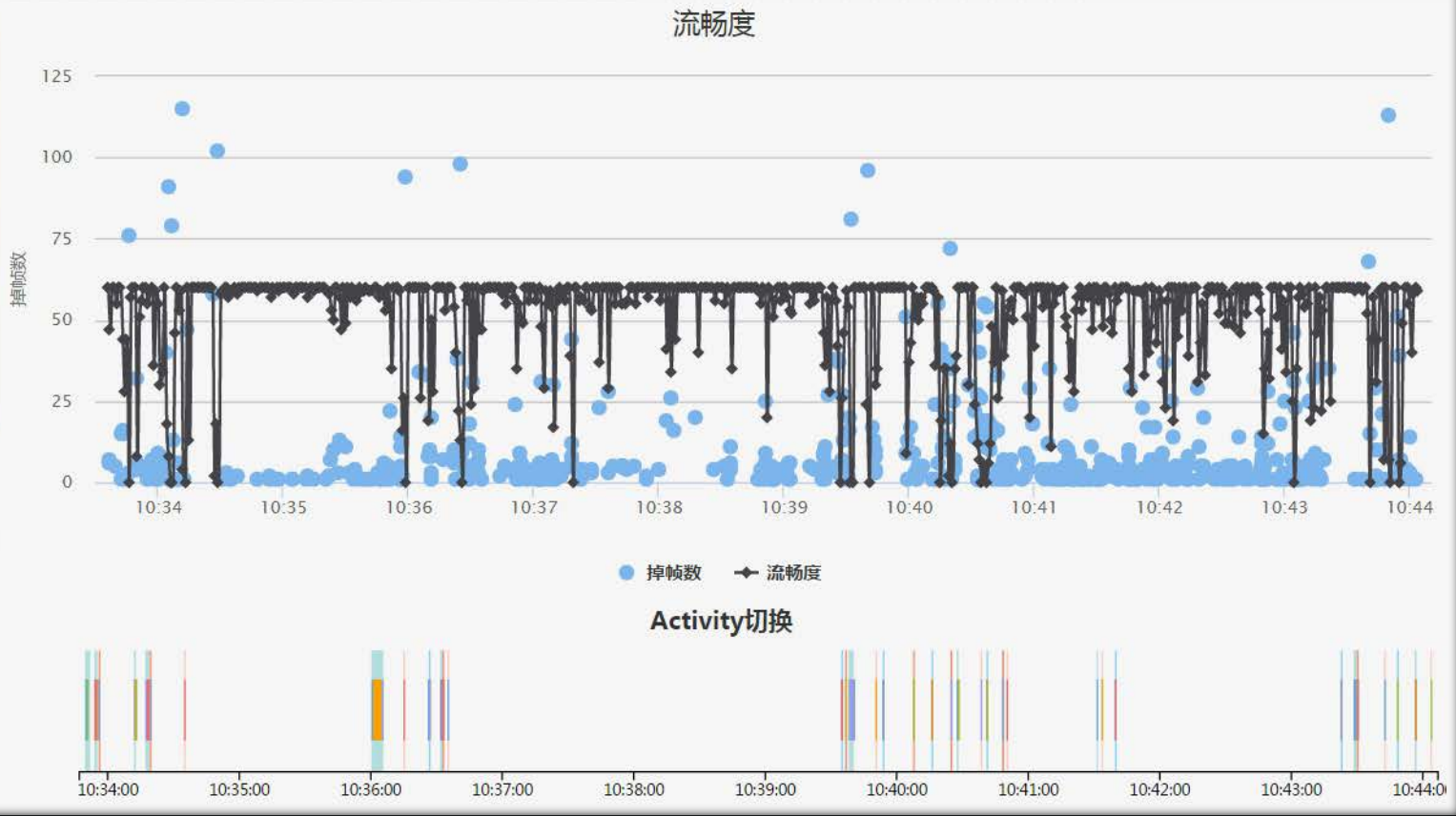
UI流畅度&卡顿

界面流畅度　　发现严重掉帧　　收起详情

机型　　操作

小米 Note 增强版 4.4.2



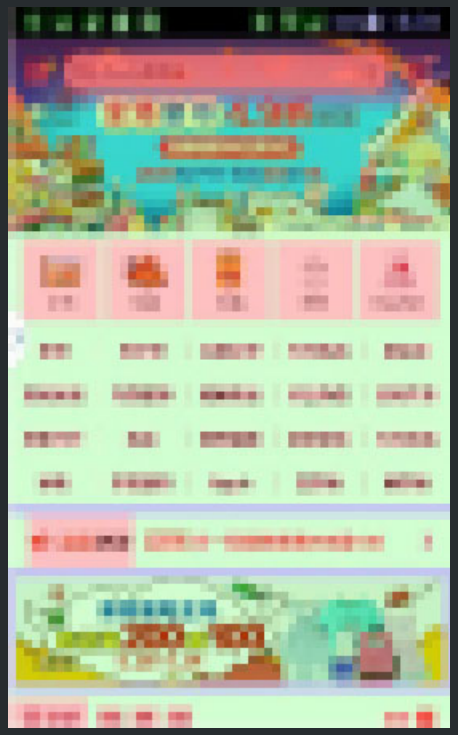


UI流畅度&卡顿


界面流畅度
发现严重掉帧
收起详情
机型
操作
小米 Note 增强版 4.4.2
流畅度
125
100
界面卡顿
出现界面卡顿
收起详情
机型
卡顿时长
基本信息
堆栈信息
操作
小米 Note 增强版 4.4.2
4574 ms
com
.java:15)
com
.java:423)
完整堆栈
0
10:34
10:35
10:36
10:37
10:38
10:39
10:40
10:41
10:42
10:43
10:44
0:00
掉帧数
流畅度
Activity切换
10:34:00
10:35:00
10:36:00
10:37:00
10:38:00
10:39:00
10:40:00
10:41:00
10:42:00
10:43:00

GPU过度绘制

无 | 1X | 2X | 3X | 4X



为什么不要过度绘制？

加重处理器负担；

增加出现掉帧的可能性。

如何减轻过度绘制？

去除无用的背景设置；

避免复杂重叠的视图；

利用ViewStub动态加载。

启动分析

Init进程 → Zygote进程 → Applications

Deamons

Services

ServiceManager

...

...

startActivity

pauseActivity

attachApplicaiton

launchActivity

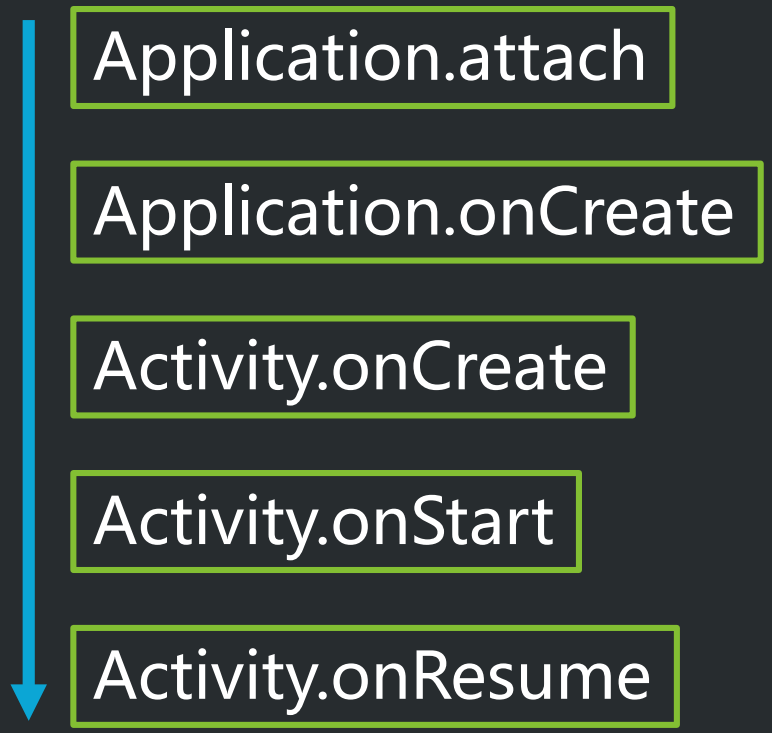
ActivityManagerService

ActivityStack

ActivityThread

ApplicationThread

启动分析



冷启动

当启动应用时，后台没有该应用的进程，这时系统会重新创建一个新的进程分配给该应用，然后启动Activity。

热启动

当启动应用时，后台已有该应用的进程，这时会从已有的进程来启动Activity。

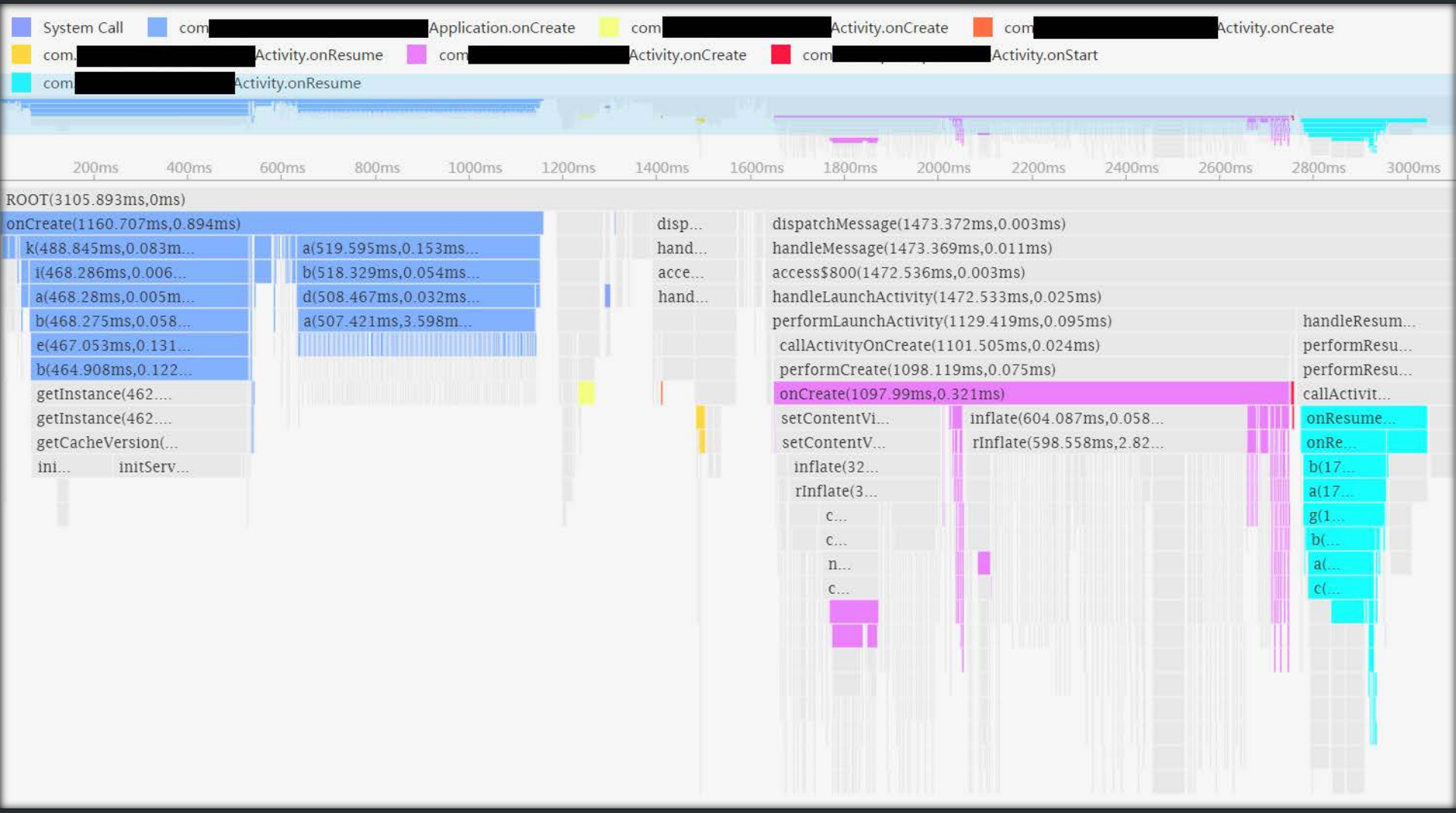
启动分析

TraceView

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
124 com[redacted]Loader.load (Ljava	3.0%	15.227	2.9%	14.891	1.9%	15.591	1.8%	14.890	1+0	15.227	15.591
Parents											
123 com[redacted].a (Land	100.0%	15.227			100.0%	15.591			1/1		
Children											
self	97.8%	14.891			95.5%	14.890					
86 dalvik/system/VMDebug.startClassPrep (	2.2%	0.336			2.1%	0.333			1/289		
(context switch)	0.0%	0.000			2.4%	0.368			1/127		
125 dalvik/system/BaseDexClassLoader.findClass	2.8%	14.318	0.1%	0.666	1.8%	15.072	0.1%	0.673	54+8	0.231	0.243
126 java/lang/String._getChars (II[CI)V	2.8%	14.142	1.7%	8.852	2.6%	21.443	1.1%	8.773	1775+0	0.008	0.012
127 java/lang/System.arraycopy (Ljava/lang/Obje	2.5%	12.697	2.5%	12.697	1.6%	13.492	1.6%	12.940	4250+0	0.003	0.003
128 com/android/internal/util/FastPrintWriter.ap	2.5%	12.642	0.8%	3.942	1.6%	12.925	0.5%	3.941	756+0	0.017	0.017
129 com[redacted]Helper.invokeIns	2.5%	12.415	0.0%	0.017	1.6%	13.132	0.0%	0.017	3+0	4.138	4.377
130 java/util/HashMap.put (Ljava/lang/Object;Lja	2.4%	12.338	0.7%	3.795	1.5%	12.510	0.5%	3.907	447+0	0.028	0.028
131 org[redacted]XmlParser.readEntity (Ljava/la	2.4%	12.180	0.6%	2.937	1.6%	12.929	0.4%	2.920	62+0	0.196	0.209
132 java/lang/StringBuilder.append (I)Ljava/lang/	2.3%	11.520	0.2%	1.112	1.4%	11.813	0.1%	1.106	232+0	0.050	0.051
133 android/content/res/XResources.getDrawabl	2.2%	11.217	0.0%	0.023	1.4%	11.424	0.0%	0.022	1+2	3.739	3.808
134 android/content/res/Resources.getDrawable	2.2%	11.206	0.0%	0.029	1.4%	11.413	0.0%	0.029	1+2	3.735	3.804
135 android/view/LayoutInflater.inflate (ILandroic	2.2%	11.184	0.0%	0.004	1.4%	11.269	0.0%	0.005	1+0	11.184	11.269

Methods

启动分析


System Call
com[redacted]Application.onCreate
com[redacted]Activity.onCreate
com[redacted]Activity.onCreate
com.[redacted]Activity.onResume
com[redacted]Activity.onCreate
com[redacted]Activity.onStart
com[redacted]Activity.onResume
200ms 400ms 600ms 800ms 1000ms 1200ms 1400ms 1600ms 1800ms 2000ms 2200ms 2400ms 2600ms 2800ms 3000ms
ROOT(3105.893ms,0ms)
onCreate(1160.707ms,0.894ms)
k(488.845ms,0.083m...
i(468.286ms,0.006...
a(468.28ms,0.005m...
b(468.275ms,0.058...
e(467.053ms,0.131...
b(464.908ms,0.122...
getInstance(462....
getInstance(462....
getCacheVersion(...
ini...
initServ...
a(519.595ms,0.153ms...
b(518.329ms,0.054ms...
d(508.467ms,0.032ms...
a(507.421ms,3.598m...
disp...
hand...
acce...
hand...
dispatchMessage(1473.372ms,0.003ms)
handleMessage(1473.369ms,0.011ms)
access$800(1472.536ms,0.003ms)
handleLaunchActivity(1472.533ms,0.025ms)
performLaunchActivity(1129.419ms,0.095ms)
callActivityOnCreate(1101.505ms,0.024ms)
performCreate(1098.119ms,0.075ms)
onCreate(1097.99ms,0.321ms)
setContentVi...
setContentV...
inflate(32...
rInflate(3...
c...
c...
n...
c...
inflate(604.087ms,0.058...
rInflate(598.558ms,2.82...
handleResum...
performResu...
performResu...
callActivit...
onResume...
onRe...
b(17...
a(17...
g(1...
b(...
a(...
c(...

启动分析

机型	冷启动时间	热启动时间	操作
小米 Note 增强版 4.4.2	4211ms	0ms	下载trace文件

创建进程：500 ms

初始化Application：508 ms

方法名	耗时(ms)	相关信息
Application.attach		问题:
[redacted]MyApplication.onCreate		长耗时方法排名 1. com[redacted].b.a(InitSystem.java:29) 2. com[redacted].java:135) 3. com[redacted].d.k(Unknown) 4. com[redacted].i.i(Unknown) 5. com.d.b.a.d([redacted]java:132)

初始化Activity [1]: ms

初始化Activity [2]: ms

方法名	耗时(ms)	相关信息
com[redacted]Activity.onCreate		长耗时方法排名 无长耗时方法
android.app.Activity.onStart		长耗时方法排名 无长耗时方法
com[redacted]Activity.onResume		长耗时方法排名 无长耗时方法

初始化Activity [3]: ms

严苛模式StrictMode

- StrictMode常用于捕获在应用主线程中进行的磁盘读写操作和网络请求；
- 当应用中有继承了Closeable接口的对象没有关闭的时候，例如文件流等，或者没有使用HTTPS进行网络请求，或者同一个Activity的实例太多，StrictMode都会给出提示。

a. 应用在主线程中进行磁盘读写；
b. 应用在主线程中进行网络请求；
c. 主线程中的某些方法的执行时间比较长；
d. SQL Cursor对象在使用之后没有关闭；
e. 继承了Closeable接口的对象在使用之后没有关闭；
f. 某一Activity有较多的实例；
g. 文件读取接口暴露给外部应用；
h. 注册某些对象(广播接收器、观察者、Listener等)后没有取消注册；
i. 没有使用加密网络(HTTPS)进行网络数据传输。

严苛模式StrictMode

```
StrictMode.setThreadPolicy(
    new StrictMode.ThreadPolicy.Builder()
    .detectCustomSlowCalls()
    .detectDiskReads()
    .detectDiskWrites()
    .detectNetwork() // or .detectAll()
    .penaltyDialog() //弹出违规提示对话框
    .penaltyLog() //在Logcat 中打印违规异常信息
    .build());
StrictMode.setVmPolicy(
    new StrictMode.VmPolicy.Builder()
    .detectLeakedSqlLiteObjects()
    .detectLeakedClosableObjects()
    .penaltyLog()
    .penaltyDeath()
    .build());
```

严苛模式StrictMode

主线程IO　　发现主线程IO　　收起详情

机型	问题类型	最长阻塞耗时	关键信息	出现次数	操作
小米 Note 增强版 4.4.2	磁盘读出	2356 ms	StrictMode policy violation; ~duration=398 ms: android.os.StrictMode$StrictModeDiskReadViolation: policy=31 violation=2	62	日志信息 下载日志

```
StrictMode policy violation; ~duration=2356 ms: android.os.StrictMode$StrictModeDiskReadViolation: policy=31 violation=2
        at android.os.StrictMode$AndroidBlockGuardPolicy.onReadFromDisk(StrictMode.java:1149)
        at libcore.io.BlockGuardOs.open(BlockGuardOs.java:106)
        at libcore.io.IoBridge.open(IoBridge.java:437)
        at java.io.FileInputStream.<init>(FileInputStream.java:78)
        at java.io.FileInputStream.<init>(FileInputStream.java:105)
        at com.stephen.performance.MainActivity.readFile(MainActivity.java:82)
        at com.stephen.performance.MainActivity.access$0(MainActivity.java:79)
        at com.stephen.performance.MainActivity$1.onClick(MainActivity.java:49)
        at android.os.Handler.handleCallback(Handler.java:808)
        at android.os.Handler.dispatchMessage(Handler.java:103)
        at android.os.Looper.loop(Looper.java:193)
        at java.lang.reflect.Method.invokeNative(Native Method)
        at java.lang.reflect.Method.invoke(Method.java:515)
```

Kyligence
Unleash big data productivity

IT大咖说
知识分享平台

谢 谢