Android内存泄漏自动化链路分析组件——Probe

张毅然 / 美团点评高级工程师

1.背景

2.业内解决方案

3.问题和策略

4.案例

5.总结

背景

- 内存溢出(OutOfMemory)复现困难
- 堆栈信息不能看出内存泄漏的根本原因

```
java.lang.OutOfMemoryError: Failed to allocate a 58 byte allocation with 5174500 free bytes and 4MB until OOM; faile
        at java.lang.AbstractStringBuilder.<init>(AbstractStringBuilder.java:89)
        at java.lang.StringBuilder.<init>(StringBuilder.java:95)
        at com.meituan.android.common.performance.statistics.anr.AnrWatchDog.run(ProGuard:174)
```

- 特别是第三方SDK的内存问题更为棘手
- 无法有效获得线上内存泄漏的可疑对象

1.背景

2.业内解决方案

3.问题和策略

4.案例

5.总结

业内解决方案

方案	针对所有内存溢出case	适用于线上环境	自动化	是否提供泄漏点路径信息
Leakcanary	否	否	是	是
MAT分析	是	是	否	是
预设可怀疑对象方案	否	否	是	是

目标

- 适用于线上app，分析线上OOM问题
- 所有的case均能检测分析
- 分析时间少
- 分析进程内存空间占用低，分析进程自己不OOM

1.背景

2.业内解决方案

3.问题和策略

4.案例

5.总结

问题和策略

- OOM时候dump内存

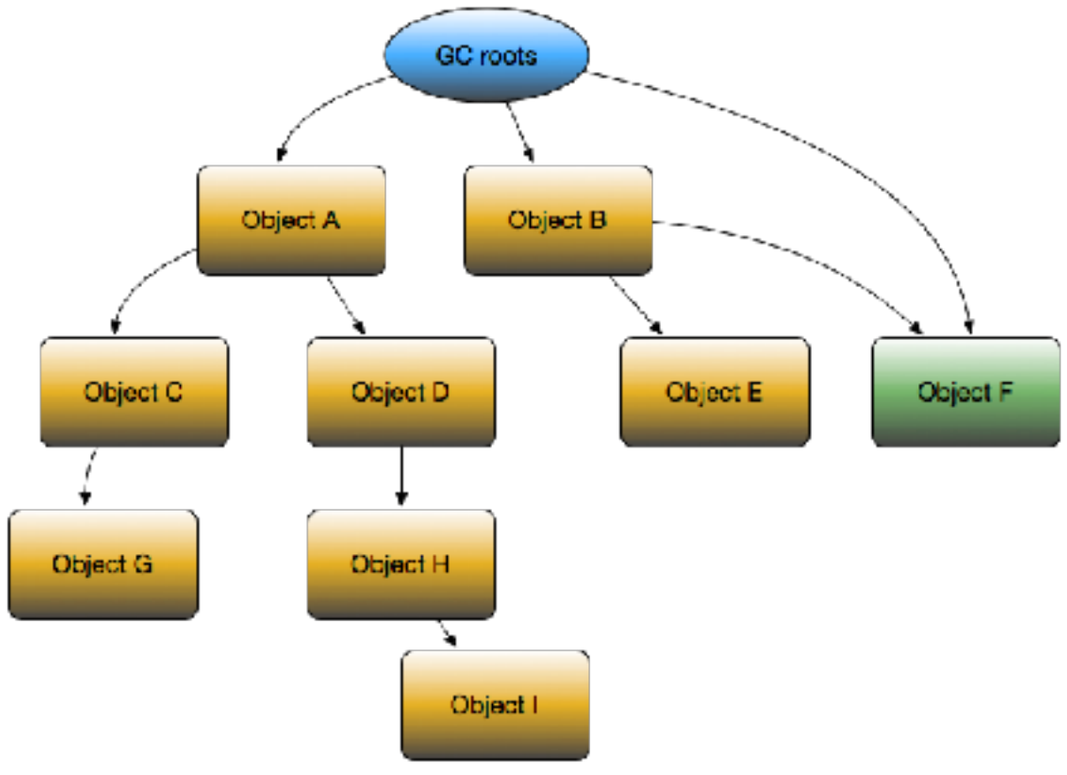
- App启动时候，单独开启进程分析

问题和策略

- 问题1：链路分析时间过长
- 问题2：分析进程占用内存过大
- 问题3：基础类型泄漏检测不到

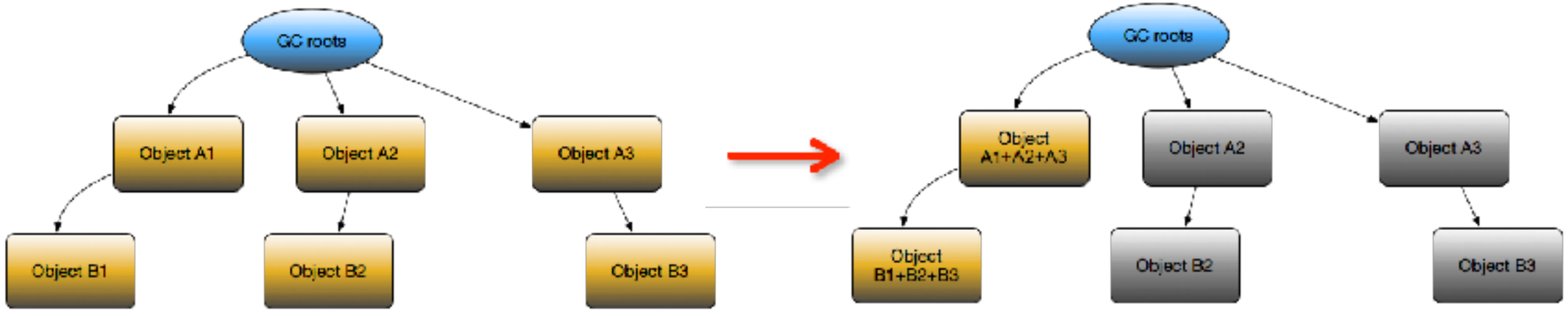
Dominator & Shallow Size & Retain Size

- Dominator 支配者 —— 如果从GC root到达对象Y的路径上，必须经过对象X，那么X就是Y的支配者。



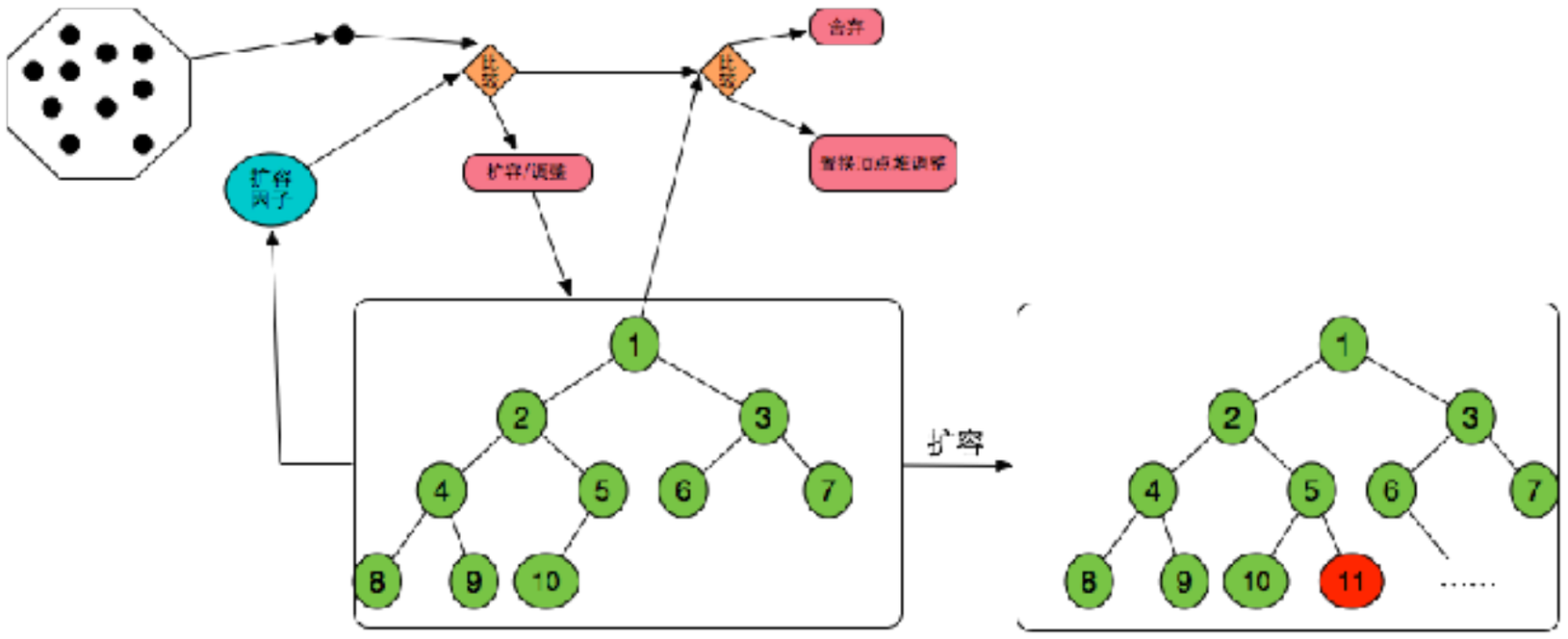
查找可疑对象

- 链路归并



查找可疑对象

- 自适应扩容法



问题和策略

- 问题1：链路分析时间过长
- 问题2：分析进程占用内存过大
- 问题3：基础类型泄漏检测不到

核心问题

分析占用内存为什么这么大?

核心问题

分析占用内存为什么这么大?

内存快照文件的大小正相关?

对比实验

- 一个实验——在一个256M阈值OOM的手机上，让一个成员变量申请特别大的一块内存200多M

- 人造OOM，dump内存，分析

- 内存快照文件达到250多M，分析进程占用内存并不大 70M左右

对比实验

- 另一个实验——在一个256M阈值OOM的手机上，添加特别多200万个小对象(72字节)

- 人造OOM，dump内存，分析

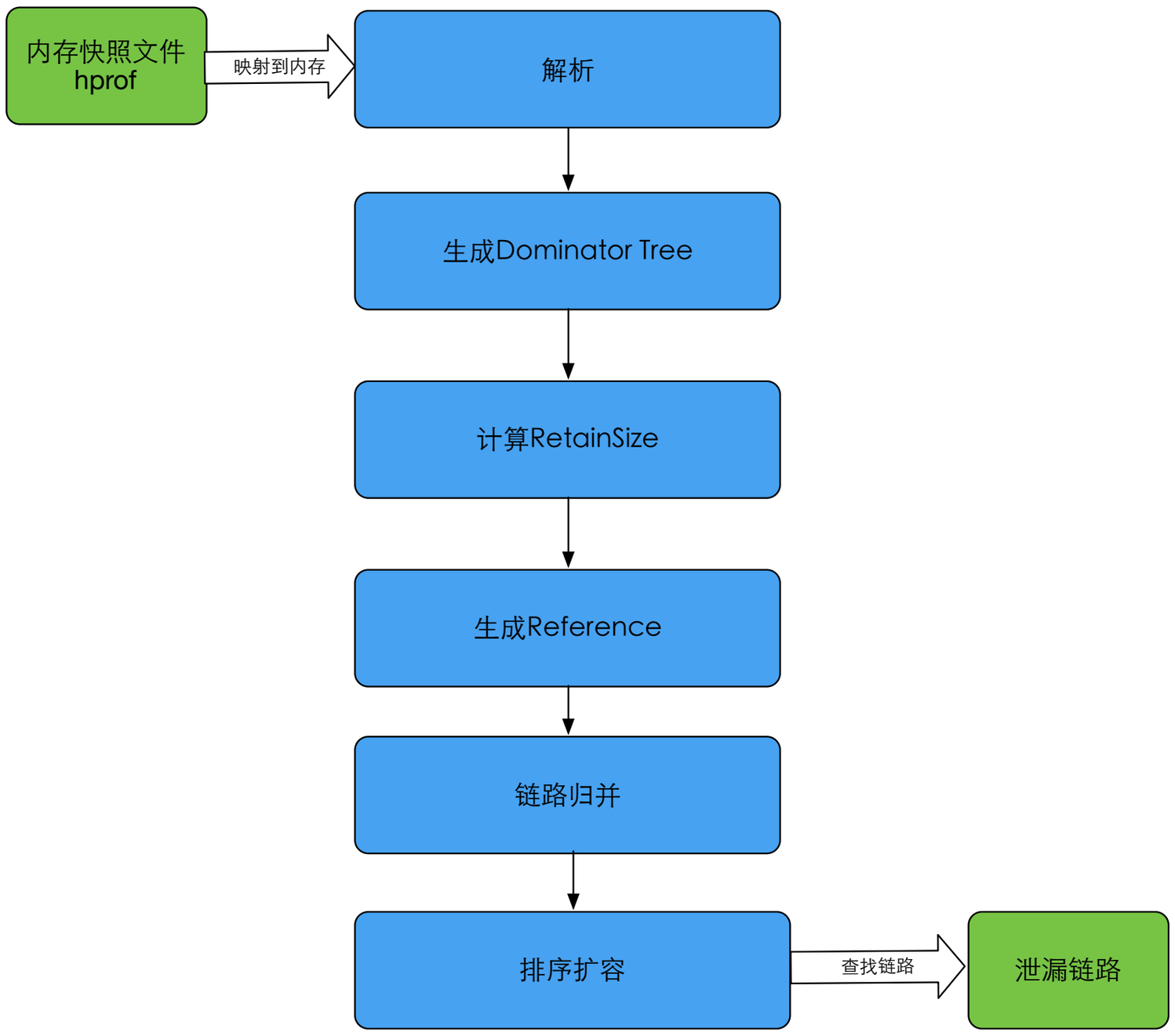
- 内存快照文件达到250多M，分析进程占用内存增长很快，在解析就OOM了

核心问题

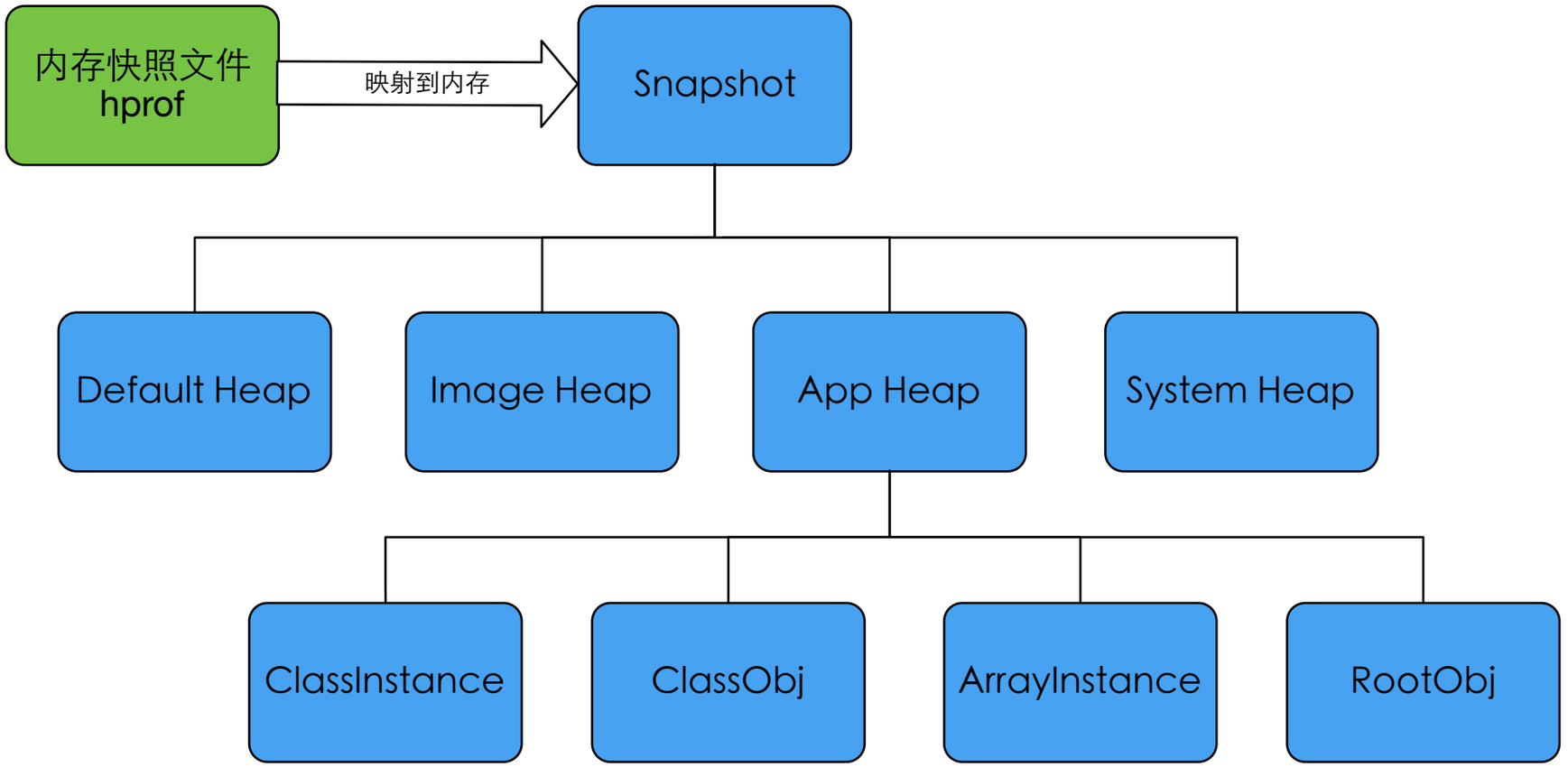
分析占用内存为什么这么大?

内存快照文件的Instance数量正相关!

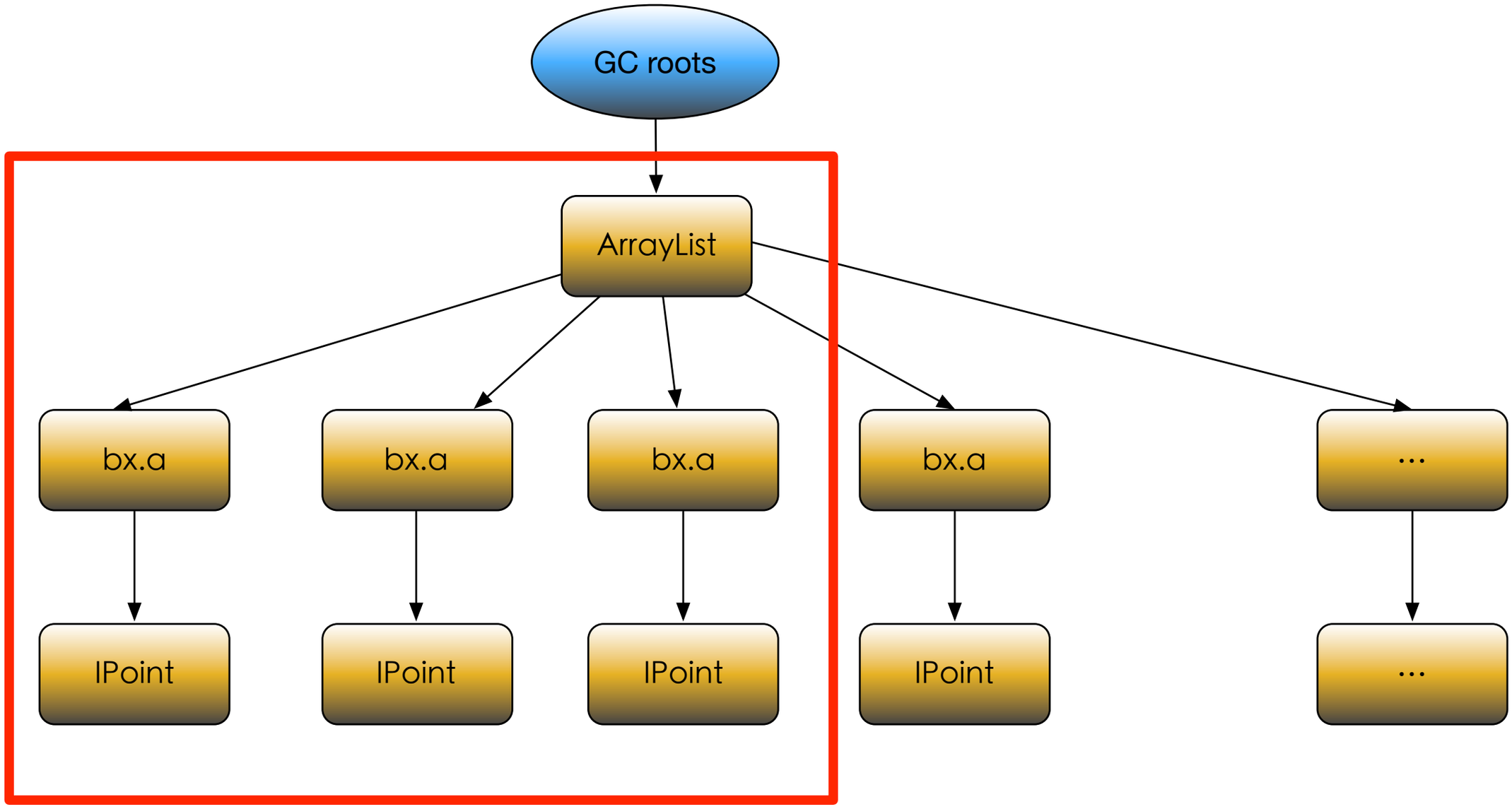
原始流程



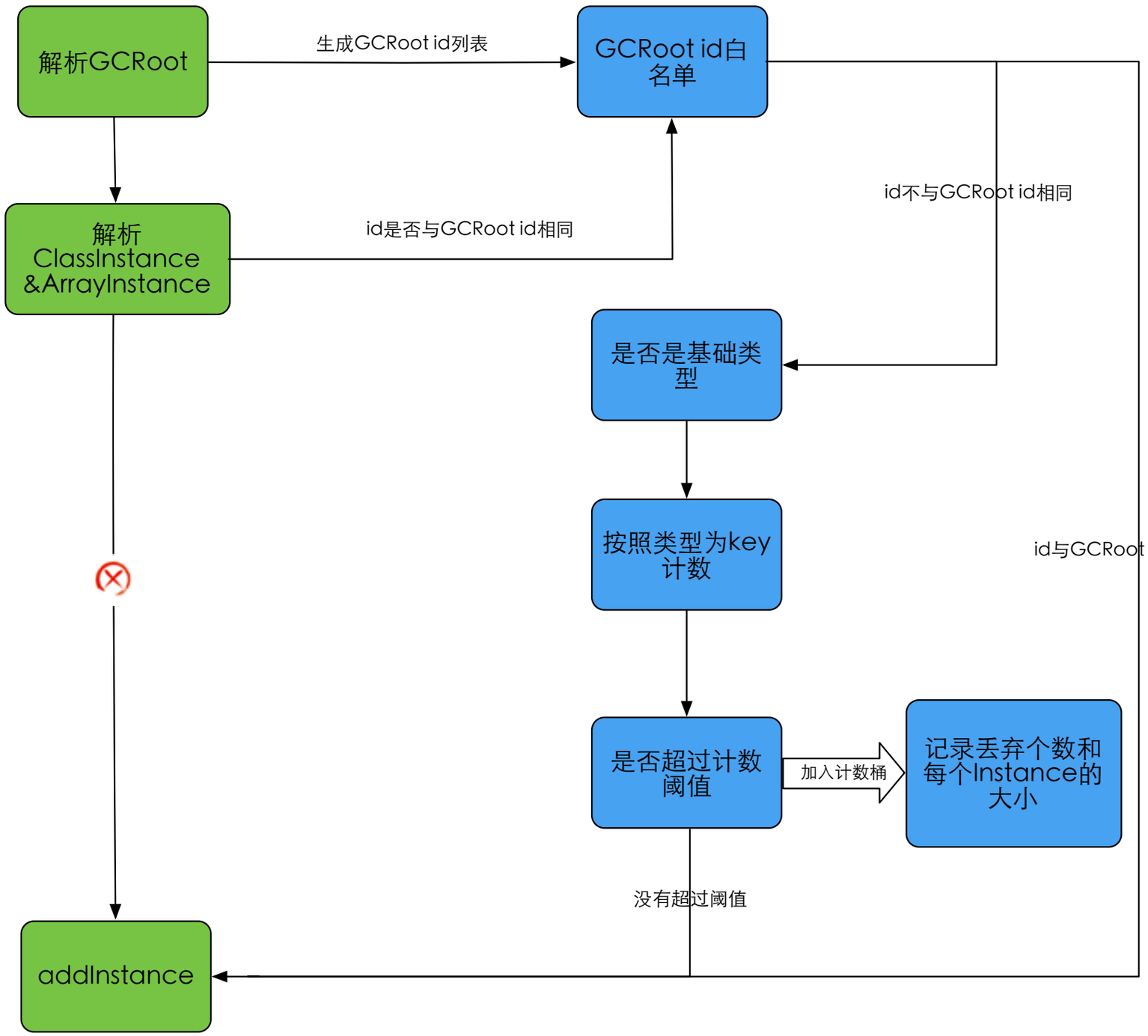
内存快照文件解析



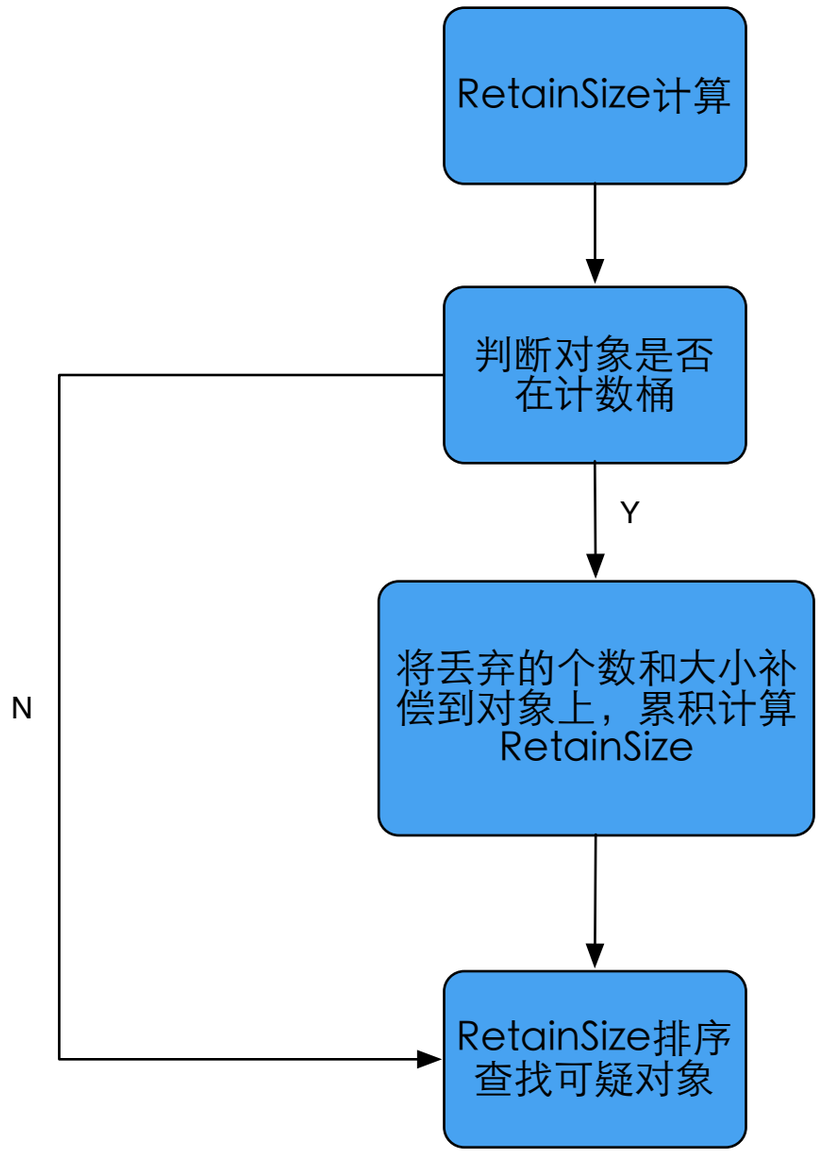
计数压缩策略



计数压缩策略



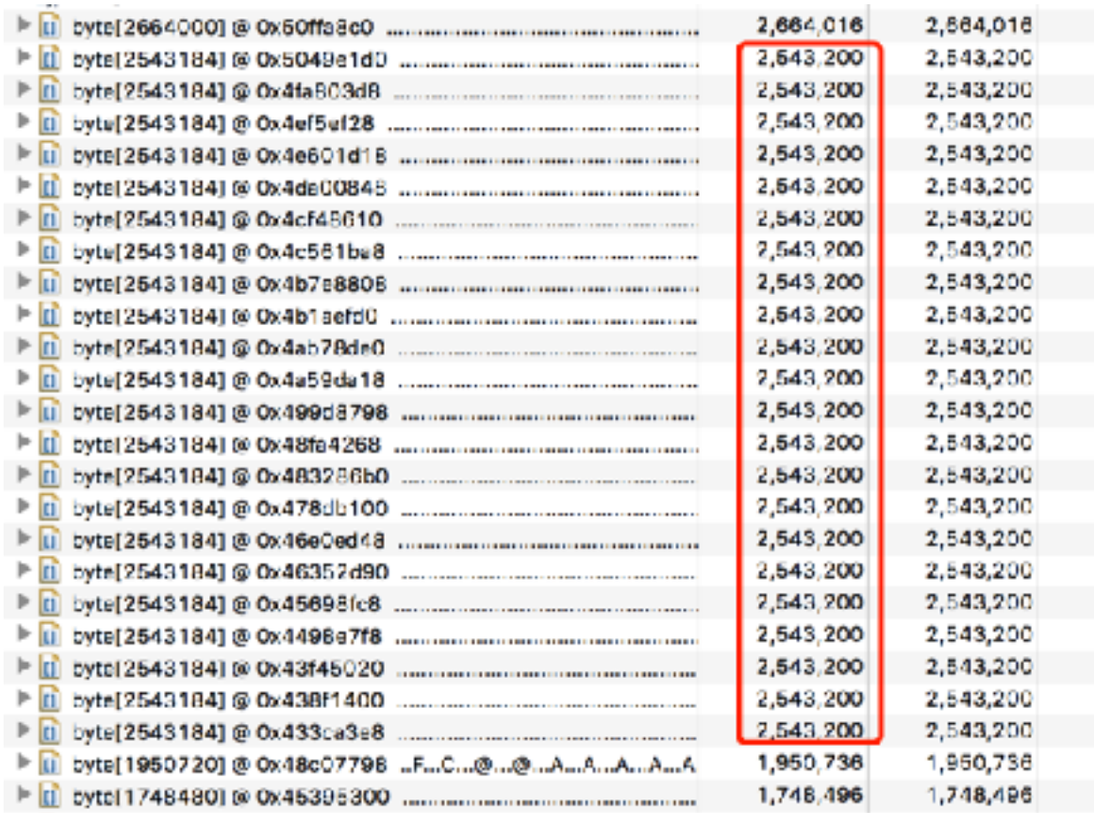
计数补偿策略



问题和策略

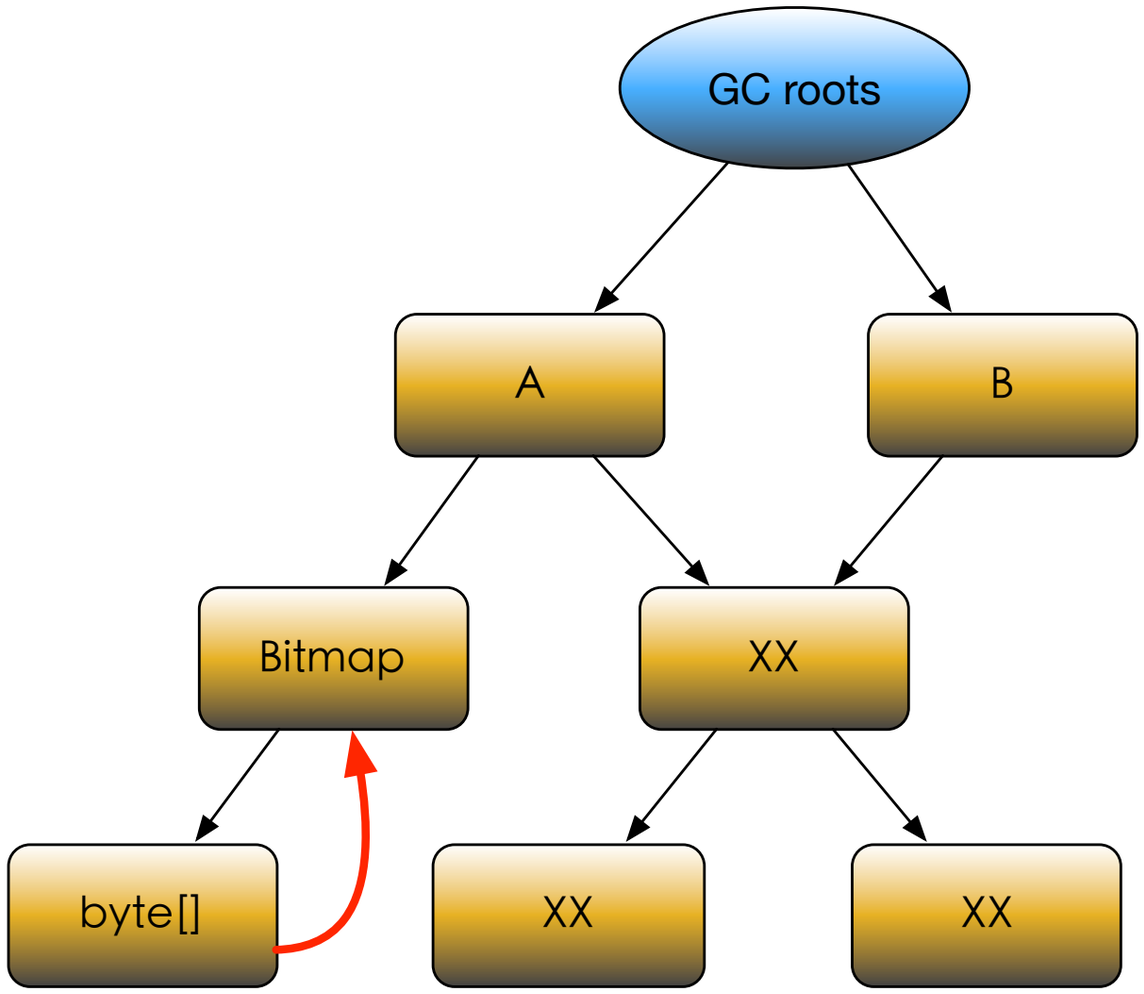
- 问题1：链路分析时间过长
- 问题2：分析进程占用内存过大
- 问题3：基础类型泄漏检测不到

Bitmap泄漏案例

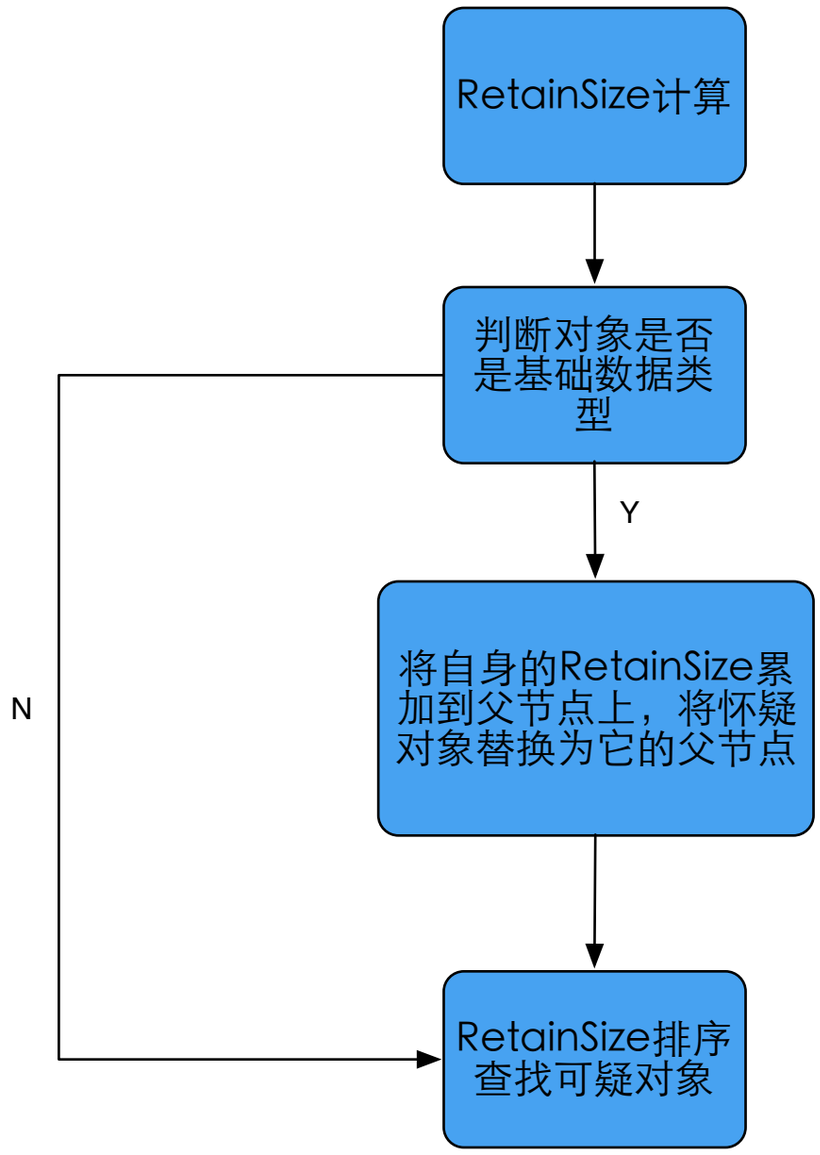


大量的Activity里面带着大量的Bitmap

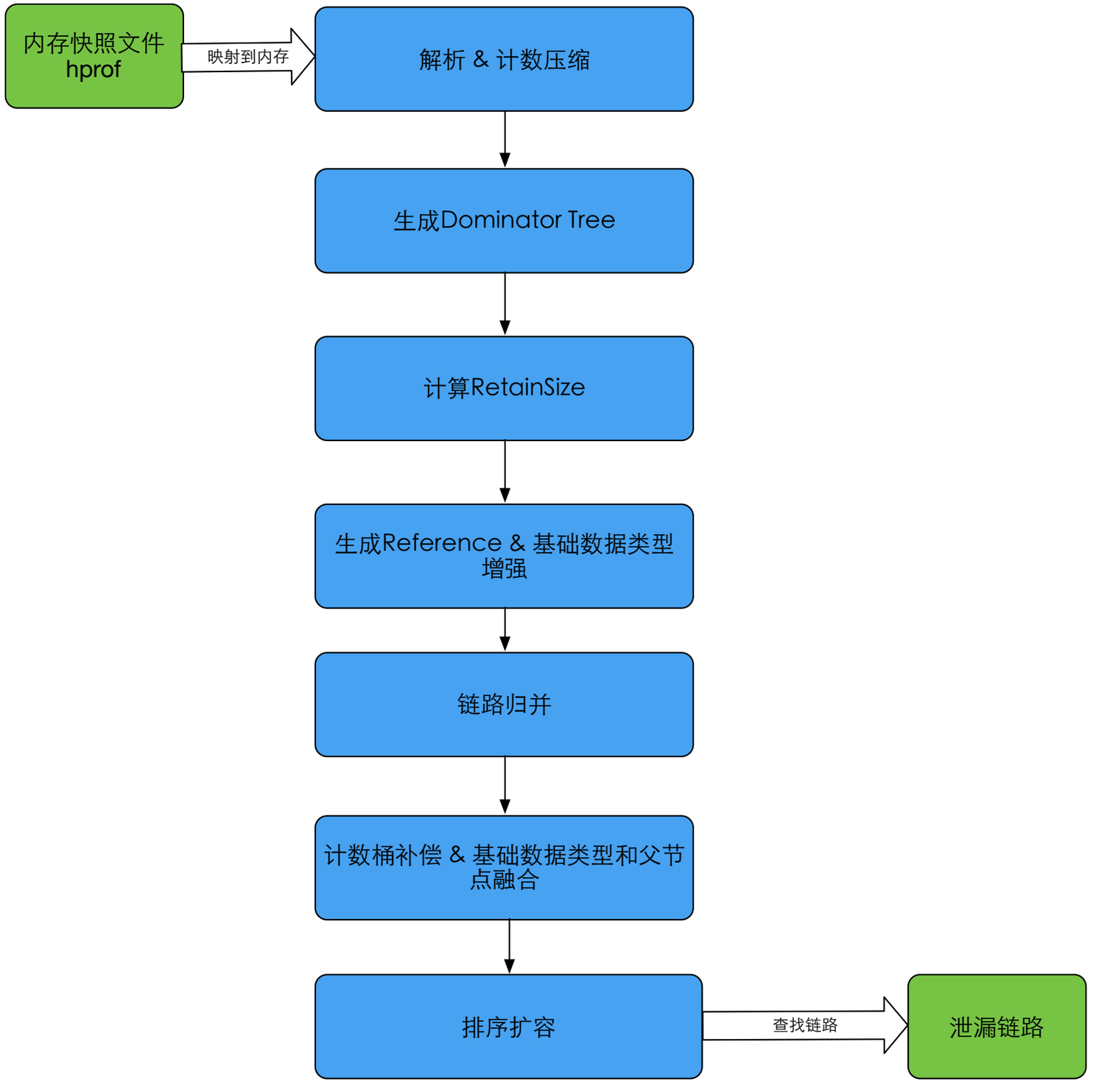
基础数据类型增强



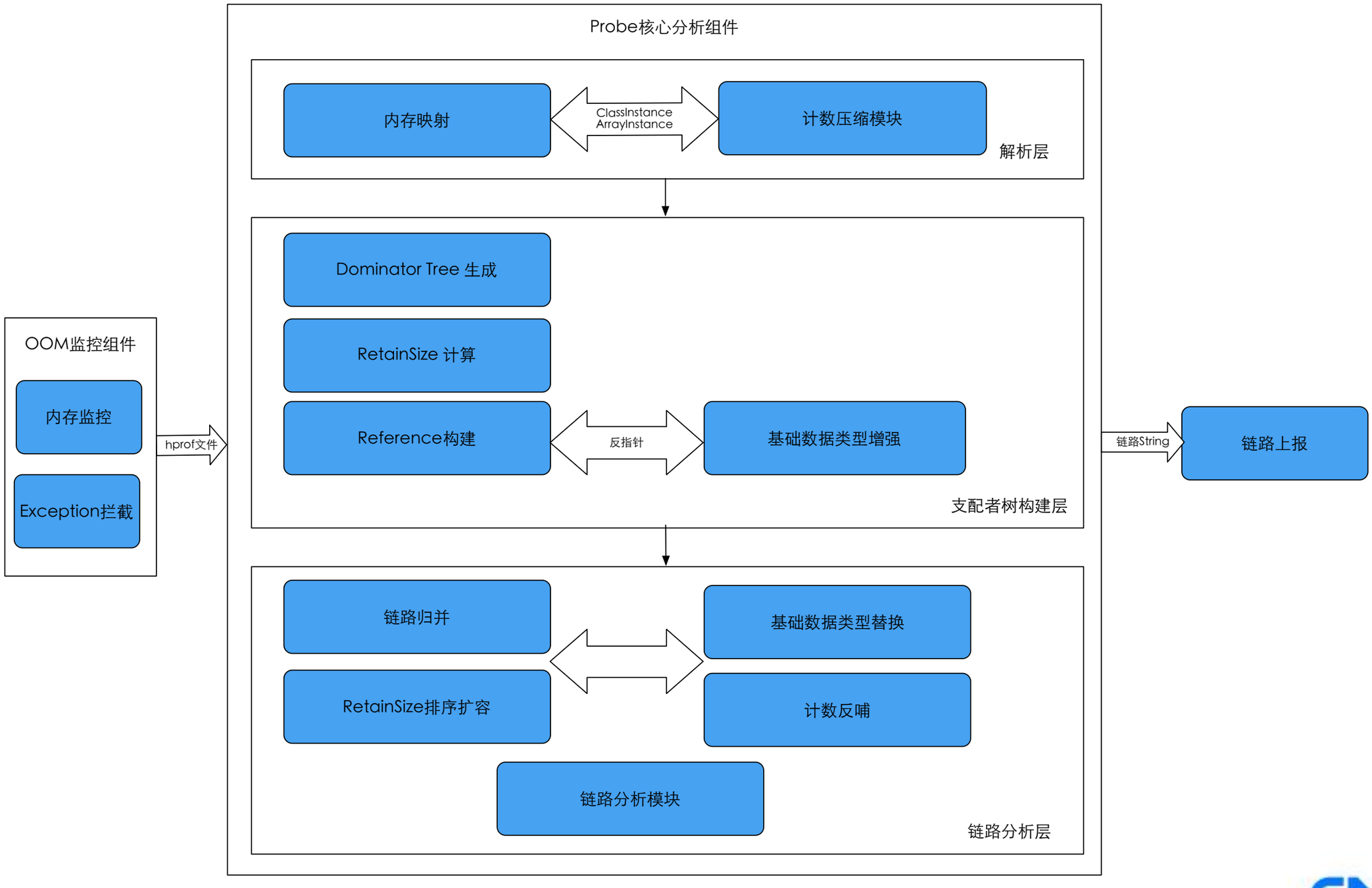
基础数据类型增强



Probe分析流程



整体结构图



1.背景

2.业内解决方案

3.问题和策略

4.案例

5.总结

案例1 Timer泄漏

```
* instance num: 114
* GC ROOT java.util.Timer$TimerImpl.<Java Local>
* references com.meituan[redacted]$2.a (anonymous subclass of java.util.TimerTask)
* references com.meituan.[redacted].a
* references android.widget.ListView.mRecycler
* references android.widget.AbsListView$RecycleBin.mCurrentScrap
* references java.util.ArrayList.array
* references array java.lang.Object[].[0]
* references com.meituan.[redacted].o
* leaks android.support.v7.widget.AppCompatImageView instance
retainSize:64706,
```

64706*114 = 7.04MB

```
* instance num: 4
* GC ROOT java.util.Timer$TimerImpl.<Java Local>
* references com.meituan.[redacted]$2.a (anonymous subclass of java.util.TimerTask)
* references com.meituan.[redacted].a
* leaks android.widget.ListView instance
retainSize:595634,
```

595634*4 = 2.27MB

案例2 地图没有正确destroy

```
* GC ROOT android.app.LoadedApk$ServiceDispatcher$DeathMonitor.this$0
* references android.app.LoadedApk$ServiceDispatcher.mConnection
* references com.loc.b.a
* references com.loc.a.d
* references java.util.ArrayList.array
* references array java.lang.Object[].[0]
* references com.meituan.[redacted].mContext
* references com.meituan.[redacted]View
* leaks com.meituan.[redacted]View instance
retainSize:188190,
* GC ROOT android.app.LoadedApk$ServiceDispatcher$DeathMonitor.this$0
* references android.app.LoadedApk$ServiceDispatcher.mConnection
* references com.loc.b.a
* references com.loc.a.d
* references java.util.ArrayList.array
* references array java.lang.Object[].[0]
* references com.meituan.[redacted].mContext
* references com.meituan.[redacted]View
* references com.meituan.[redacted]View
* leaks android.widget.RelativeLayout instance
retainSize:3414,
```

案例3 某定位SDK内存泄漏

分析进程内存占用最高大约100M

```
[dump hprof fileLength 457.8916244506836 MB
, compute start:1488877087790, compute finish in:16s
, suspiciousMTInstances size: 11
, leak trace start:1488877104798
,
* GC ROOT [redacted]mNmeaBuffer
* leaks java.util.ArrayList MTInstance
retainSize:4884482,
* MTInstance num: 1982875
* GC ROOT [redacted]mNmeaBuffer
* references java.util.ArrayList.array
* references array java.lang.Object[].[11329]
* leaks [redacted]$Nmea MTInstance
retainSize:24,
* MTInstance num: 1997729
* GC ROOT java.lang.String MTInstance
retainSize:84,
**********
AbandonedInstance char[] size=0 count=1989267
AbandonedInstance [redacted]$Nmea size=24 count=1964285
AbandonedInstance java.lang.String size=16 count=1984487
,
leak trace finish in:122s
]
```

案例4 Bitmap泄漏

```
analysis start: 2017.03.21.07.42.23
[dump hprof fileLength 521.9818143844604 MB
, compute start:1490096550299, compute finish in:20s
, suspiciousMTInstances size: 15
, leak trace start:1490096570879
,
* MTInstance num: 294
* GC ROOT static com.meituan.[redacted]activities
* references java.util.ArrayList.array
* references array java.lang.Object[].[118]
* references com.meituan.[redacted]Activity.image
* references android.widget.ImageView.mDrawable
* references android.graphics.drawable.BitmapDrawable.mBitmapState
* references android.graphics.drawable.BitmapDrawable$BitmapState.mBitmap
* leaks android.graphics.Bitmap MTInstance
retainSize:4194380,
******************************
AbandonedInstance char[] size=0 count=5325
AbandonedInstance java.lang.reflect.ArtMethod size=48 count=23381
AbandonedInstance java.lang.reflect.ArtField size=24 count=1708
AbandonedInstance java.lang.String size=24 count=1084
,
leak trace finish in:82s
]
```

案例5 某地图SDK路线绘制泄漏

某地图SDK搜索路线绘制泄漏

```
[dump hprof fileLength 288.9945297241211 MB
, compute start:1490764729787, compute finish in:26s
, suspiciousMTInstances size: 10
, leak trace start:1490764756572
```

```
* GC ROOT thread java.lang.Thread.<Java Local> (named 'Thread name not available')
* leaks java.util.ArrayList MTInstance
retainSize:14132276,
* MTInstance num: 3073513
* GC ROOT thread java.lang.Thread.<Java Local> (named 'Thread name not available')
* references java.util.ArrayList.array
* references array java.lang.Object[].[0]
* leaks [redacted].bx$a MTInstance
retainSize:56,
* MTInstance num: 3073514
* GC ROOT thread java.lang.Thread.<Java Local> (named 'Thread name not available')
* leaks [redacted].IPoint MTInstance
retainSize:16,
* MTInstance num: 1058
* GC ROOT com.android.volley.NetworkDispatcher.<Java Local>
* references com.android.volley.NetworkDispatcher.interruptActions
* leaks java.util.ArrayList MTInstance
retainSize:20,
* MTInstance num: 134
* GC ROOT thread java.lang.Thread.<Java Local> (named 'thread')
* references com.meituan.[redacted].x
* references com.meituan.[redacted].d
* references java.util.ArrayList.array
* references array java.lang.Object[].[11]
* references com.[redacted].c
* leaks android.graphics.Bitmap MTInstance
retainSize:14476,
*****************************
AbandonedInstance char[] size=0 count=5626
AbandonedInstance [redacted].IPoint size=16 count=3063506
AbandonedInstance java.lang.reflect.ArtMethod size=48 count=19779
AbandonedInstance [redacted].bx$a size=56 count=3063513
AbandonedInstance java.lang.String size=24 count=1301
,
leak trace finish in:95s
]
```

案例6 某SDK数据缓存泄漏

[dump hprof fileLength 230.62209129333496 MB
, compute start:1494324467752, compute finish in:77s
, suspiciousMTInstances size: 12
, leak trace start:1494324551821

, * MTInstance num: 512270
* GC ROOT android.os.HandlerThread.<Java Local>
* references java.util.Collections$SynchronizedMap.m
* references java.util.HashMap.table
* references array java.util.HashMap$HashMapEntry[].[159]
* references java.util.HashMap$HashMapEntry.value
* references java.util.concurrent.CopyOnWriteArrayList.elements
* references array java.lang.Object[].[0]
* references com.meituan.[redacted]$GearsInfo.wifi
* references java.util.ArrayList.array
* references array java.lang.Object[].[0]
* leaks com.meituan.[redacted]$GearsInfo$MyScanResult MTInstance
retainSize:28,
* MTInstance num: 55978
* GC ROOT android.os.HandlerThread.<Java Local>
* leaks android.location.Location MTInstance
retainSize:132,
* MTInstance num: 47523
* GC ROOT android.os.HandlerThread.<Java Local>
* leaks com.meituan.[redacted]CellInfo MTInstance
retainSize:84,
* MTInstance num: 102970
* GC ROOT com.data.carrier_v4.d$1.<Java Local> (anonymous subclass of android.os.HandlerThread)
* leaks java.util.ArrayList MTInstance
retainSize:20,
* MTInstance num: 48299
* GC ROOT android.os.HandlerThread.<Java Local>
* leaks com.meituan.[redacted]$GearsInfo MTInstance
retainSize:24,

1.背景

2.业内解决方案

3.问题和策略

4.案例

5.总结

总结

- 适用于线上环境
- 分析时间快，2min~8min
- 占用内存低，分析进程平均占用100M
- 分析成功率高，88%
- 特别适合追查三方SDK的内存问题

总结

方案	针对所有内存溢出case	适用于线上环境	自动化	是否提供泄漏点路径信息
Leakcanary	否	否	是	是
MAT分析	是	是	否	是
预设可怀疑对象方案	否	否	是	是
Probe	**是**	**是**	**是**	**是**

OutOfMemory分析组件Probe

美团点评

外卖配送技术团队

Q&A

THANKS!

GMTC
全球移动技术大会2017