Android开发中常见的内存泄漏

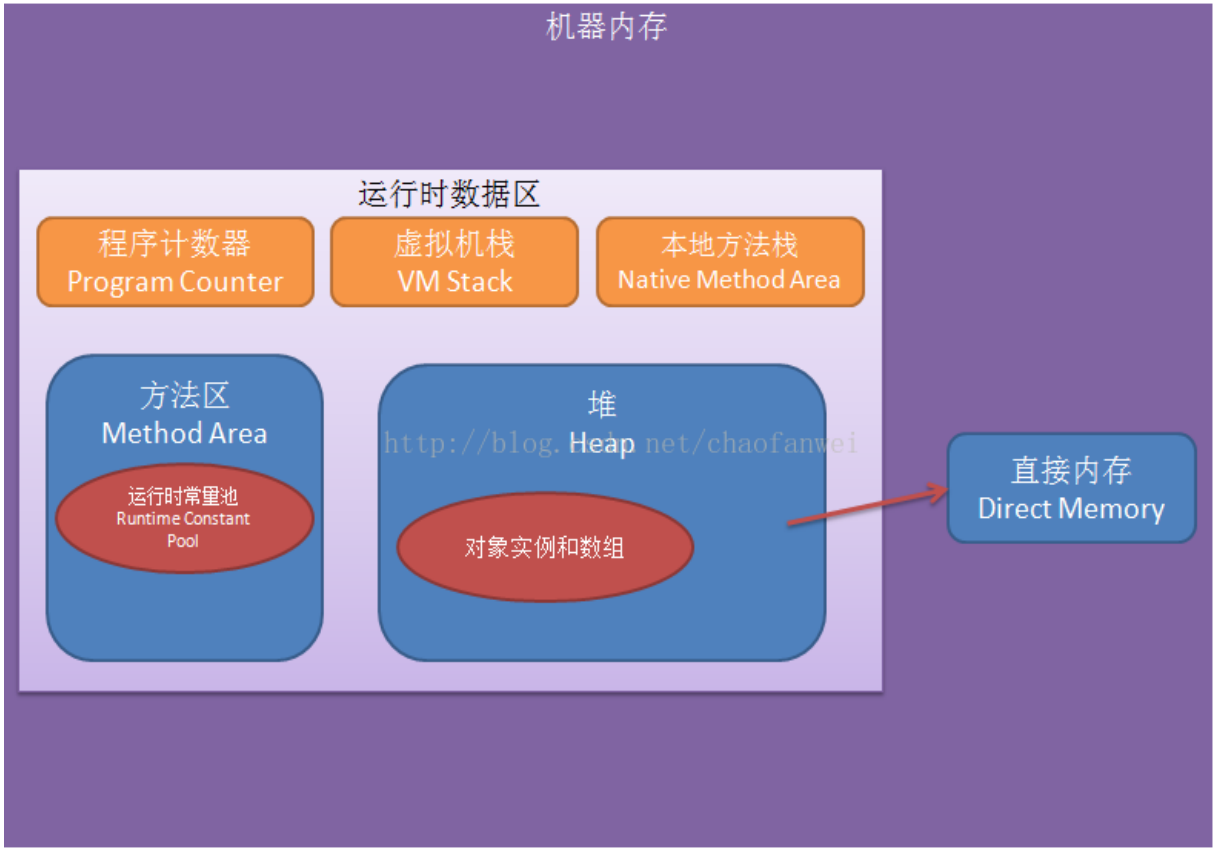
张茜婷
zhangxiting@baidu.com

概述

- Java内存结构
- Java回收机制及引用方式
- Android开发中常见内存泄漏
- 工具

Baidu 百度

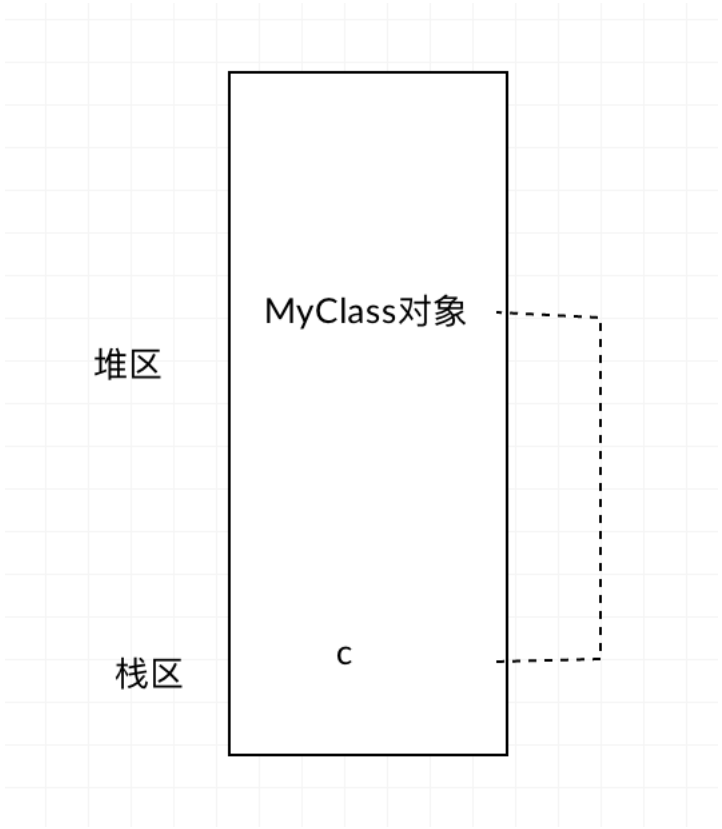
Java内存结构



Baidu 百度

堆区和栈区

```
//C++
void function()
{
        MyClass* c = new MyClass();
        c->show();

        delete c;
        c = NULL;
}
//JAVA
public void function{
        MyClass c = new MyClass();
        c.show();
}
```



Baidu 百度

Java回收机制

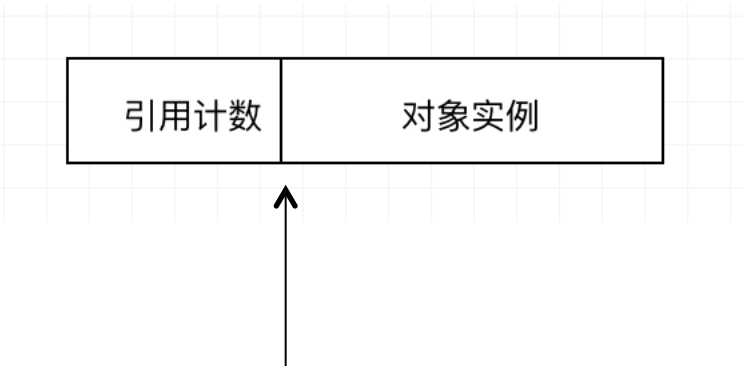
- 概念：在Java中，当某块内存没有任何对象引用时，这块内存便成为垃圾。JVM的一个系统级线程会自动释放该内存块。

- 垃圾回收算法
  - 引用计数
  - 标记-清除算法
  - 复制算法
  - 分代

Baidu 百度

垃圾回收算法-----引用计数回收法

- 原理
- 记录每个对象被引用的次数。每当创建一个新的对象，或者其他指针指向它时，计数器加一；每当指向其的指针移除时，计数器减一；当计数器降为0时，删除对象并回收内存。



垃圾回收算法-----标记清除回收

- 原理

从 “GC Roots”集合开始，将内存整个遍历一次，保留所有可以被GC Roots直接或间接引用到的对象，剩下的对象都当做垃圾被清除。

- 1）标记（mark）阶段

针对GC Roots每一个对象，采用递归的方式，处理其直接、间接引用到的所有对象

- 2）清除（sweep）阶段

遍历内存，将所有未标注的对象全部回收

并将保留的对象的标注清除，以便下次GC时使用

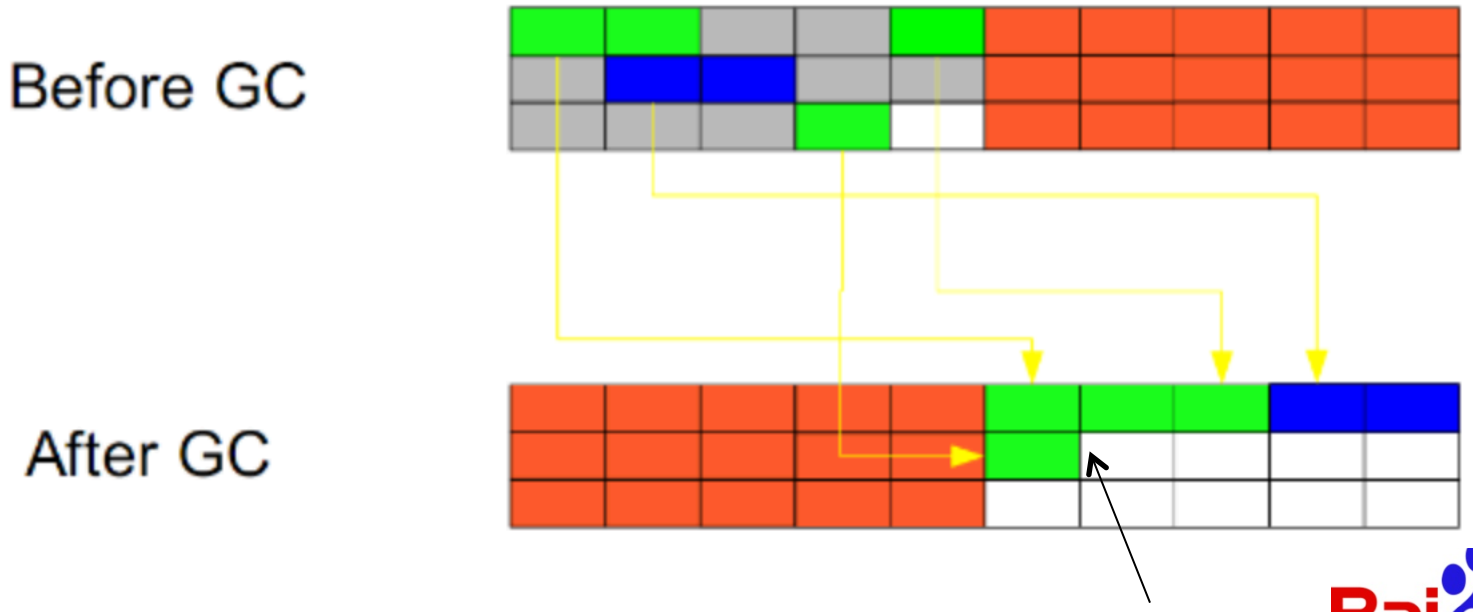
Baidu 百度

垃圾回收算法-----复制回收

- 原理

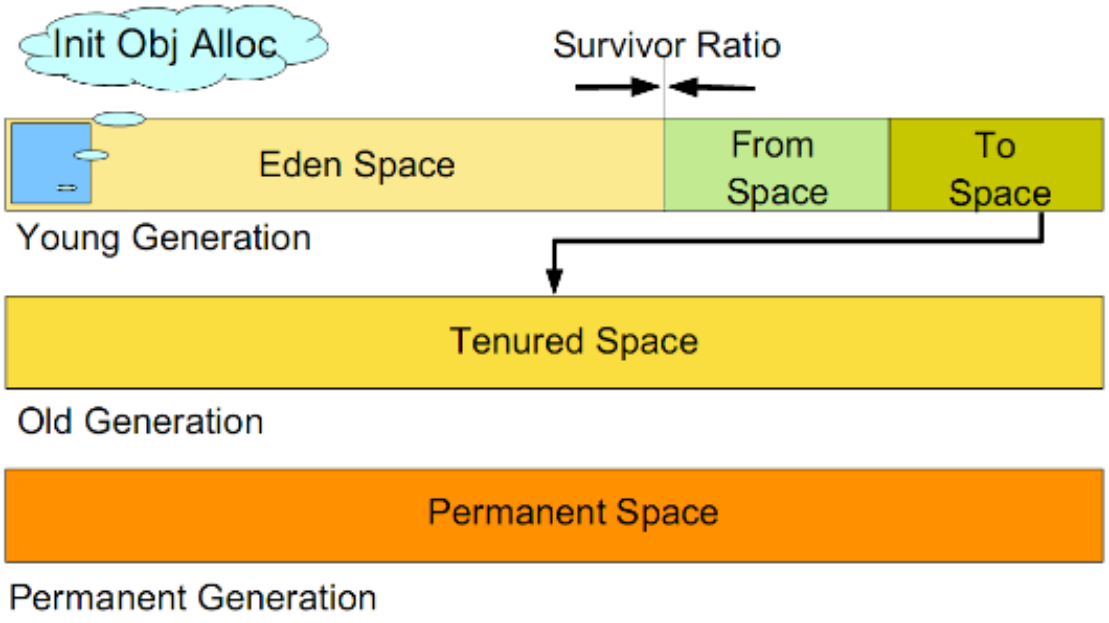
将堆分为两部分，称为『乒』『乓』。维护一个指针---下个对象分配的起始地址。

当『乒』的内存分配完毕之后，利用标记识别出存活对象，复制到『乓』，再从『乓』分配内存，之后分配完毕，复制到『乒』，依次继续。



垃圾回收算法----分代回收

- 前提
  - 大部分对象创建完很快就没用了
  - 每次GC收集的90%的对象都是上次GC后创建的
  - 如果对象可以活过一个GC周期，那么它在后续几次GC中变为垃圾的可能性很小。
- 原理



Java中的引用

Jdk1.2中引入Java.lang.ref包

⑴强引用（StrongReference）

- 垃圾回收器绝不会回收它，当内存空间不足，Java虚拟机宁愿抛出（OOM）OutOfMemory错误，使程序异常终止。

⑵软引用（SoftReference）

- 内存空间足够，垃圾回收器就不会回收它；
- 如果内存空间不足了，就会回收这些对象的内存。

⑶弱引用（WeakReference）

- 弱引用与软引用的区别在于：只具有弱引用的对象拥有更短暂的生命周期。
- 不管当前内存空间足够与否，都会回收它的内存。

⑷虚引用（PhantomReference）

- 虚引用与其他几种引用都不同，虚引用并不会决定对象的生命周期。如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。

Baidu 百度

Android中常见内存泄漏

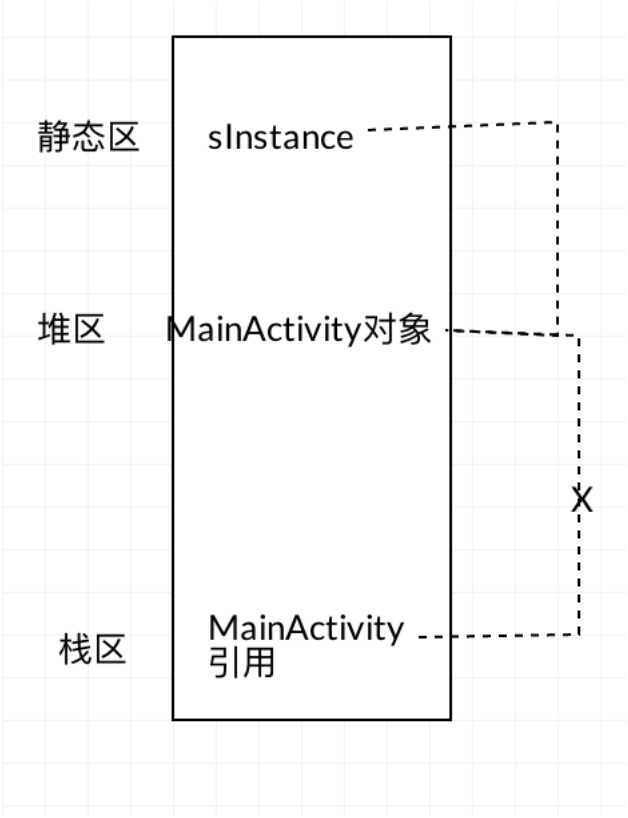
- Activity 对象持有其 View 层以及相关联的所有资源文件的引用，如果内存泄漏发生在 Activity 中，那么将损失大量的内存空间。

- 静态引用导致的内存泄漏
- 非静态内部类/匿名内部类导致的内存泄漏
- 其他

Baidu 百度

静态成员（一）--------单例模式直接引用

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



AppManager appm = AppManager.getInstance(MainAcitivity.this);

静态对象直接引用**Activity**导致内存泄漏

Baidu 百度

静态成员（二）--------间接引用

```
public class MainActivity extends Activity {
    private static Drawable mDrawable;

    @Override
    protected void onCreate(Bundle saveInstanceState) {
        super.onCreate(saveInstanceState);
        setContentView(R.layout.activity_main);
        ImageView iv = new ImageView(this);
        mDrawable = getResources().getDrawable(R.drawable.ic_launcher);
       iv.setImageDrawable(mDrawable);
    }
}
```

mDrawable -> ImageView -> MainActivity

静态对象间接引用Activity 导致内存泄漏

Baidu 百度

非静态内部类/匿名内部类（一）

- 非静态的内部类和匿名内部类都会隐式地持有其外部类的引用

```
public class MainActivity extends Activity
{
      static Demo sInstance = null;
     public void onCreate(BundlesavedInstanceState)
    {
     }
      class Demo
     {
      void do Something()
      {
      }
   }
}
```

- 静态成员是内部类对象。

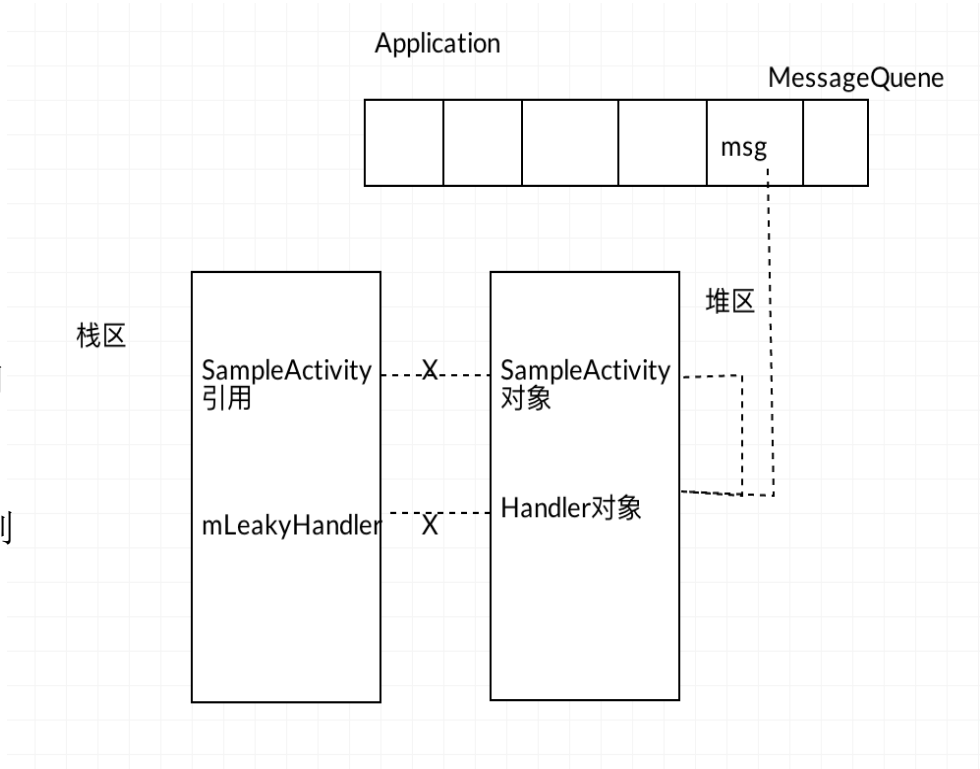
非静态内部类/匿名内部类（二）----------Android中Handler引起的内存泄漏

```
public class SampleActivity extends Activity {

    private final Handler mLeakyHandler = new Handler() {

        public void handleMessage(Message msg) {
          // ...
        }
    }
}
```

1.主线程中的Looper生命周期和当前应用一样长。

2.当一个Handler在主线程进行了初始化之后，我们发送一个target为这个Handler的消息到Looper处理的消息队列时，实际上已经发送的消息已经包含了一个Handler实例的引用，只有这样Looper在处理到这条消息时才能找到对应的handlemessage方法。

3.匿名内部类含有对外部类对象的引用



非静态内部类/匿名内部类（三）----------线程引起的内存泄漏

```
public class SampleActivity extends Activity {

    new AsyncTask<Void, Void, Void>() {
        @Override
        protected Void doInBackground(Void... params) {
          doSomething();
            return null;
        }
    }.execute();
```

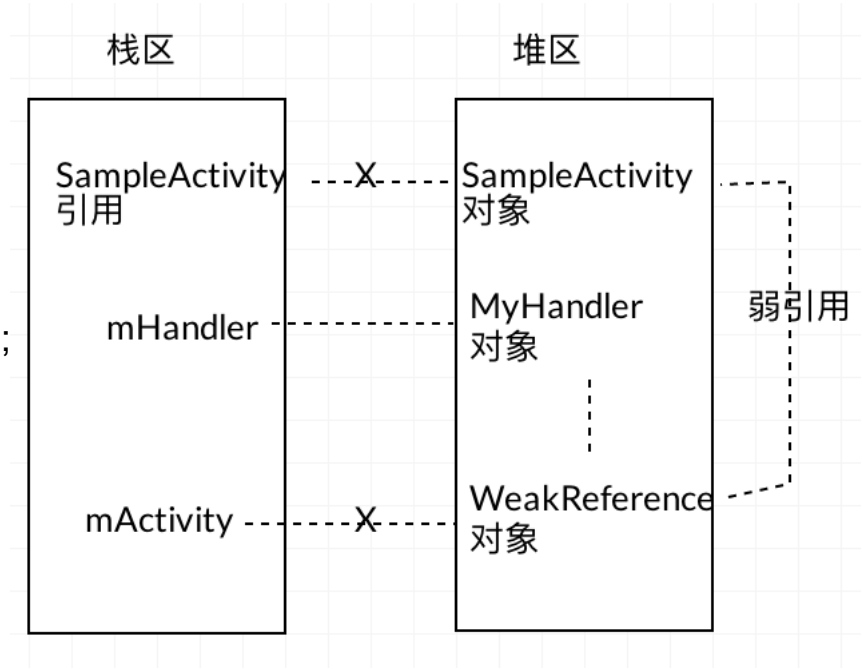
线程运行时长不定，却拥有Activity的引用

```
    new Thread(new Runnable() {
        @Override
        public void run() {
              doSomething();
             }
    }).start();
}
```

Baidu 百度

解决思路--- static内部类 + 弱引用

```
public class SampleActivity extends Activity {

   private static class MyHandler extends Handler {

        private final WeakReference<SampleActivity> mActivity;

       public MyHandler(SampleActivity activity) {
          mActivity =  new WeakReference<SampleActivity>(activity);
       }
        public void handleMessage(Message msg) {
           SampleActivity activity = mActivity.get();
          if (activity != null) {
          // ...
     }
   }

private final MyHandler mHandler = new MyHandler(this);

protected void onDestroy() {
       super.onDestroy();
       mHandler.removeCallbacksAndMessages(null);
       //或关闭线程
    }
  }
```



避免context内存泄漏的小建议

1.能使用Application的context时，尽量使用Application的context

getApplication() / getApplicationContex()

getApplication() 仅在Activity 和Service 中有效

getApplicationContext() 在Context中都有效，优先使用getApplicationContext.

2.不要让生命周期长于Activity的对象持有到Activity的引用

3.尽量不要在Activity中使用非静态内部类

使用静态内部类+弱引用

Baidu 百度

其他情况的内存泄漏

- 资源对象没关闭造成的内存泄漏

  查询数据库而没有关闭Cursor

  文件、输入输出流

  Bitmap未recycle

  WebView未destroy

- 注册某个对象后未反注册

  Register unregister

- 集合中对象没清理造成的内存泄漏

  Set HashMap clear/remove

- 构造Adapter时，没有使用缓存的 convertView

Baidu 百度

关于内存泄漏的调试工具

- 内存监测工具 DDMS --> Heap
  - Dalvik Debug Monitor Service，是 Android 开发环境中的Dalvik虚拟机调试监控服务
  - 位置：Android/sdk/tools/monitor

- 内存分析工具 MAT Memory Analyzer Tool

  由Eclips提供，分为Eclips插件版 和 独立版

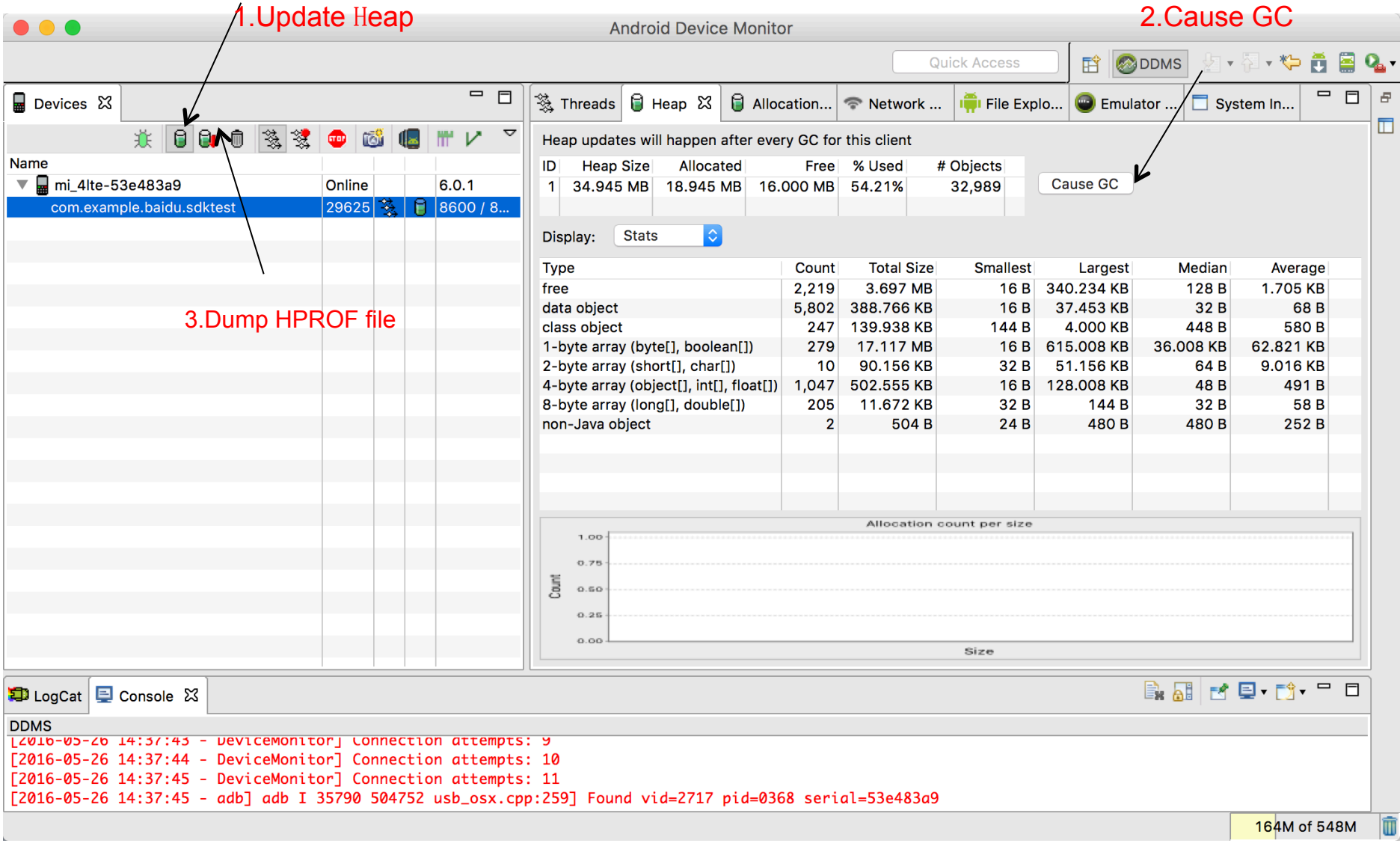
  下载地址： http://eclipse.org/mat/downloads.php

Bai du 百度

简单例子演示

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

        MapView mapView = new MapView(this,new BaiduMapOptions());
        setContentView( mapView);


        new Thread(new Runnable() {

            public void run() {
                try {
                    Thread.sleep(60 * 60 * 1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            }).start();

    }
}
```

DDMS


1.Update Heap
2.Cause GC
3.Dump HPROF file
Android Device Monitor
Quick Access
DDMS
Devices
Threads
Heap
Allocation...
Network ...
File Explo...
Emulator ...
System In...
Heap updates will happen after every GC for this client
ID Heap Size Allocated Free % Used # Objects
1 34.945 MB 18.945 MB 16.000 MB 54.21% 32,989
Cause GC
Display: Stats
Type Count Total Size Smallest Largest Median Average
free 2,219 3.697 MB 16 B 340.234 KB 128 B 1.705 KB
data object 5,802 388.766 KB 16 B 37.453 KB 32 B 68 B
class object 247 139.938 KB 144 B 4.000 KB 448 B 580 B
1-byte array (byte[], boolean[]) 279 17.117 MB 16 B 615.008 KB 36.008 KB 62.821 KB
2-byte array (short[], char[]) 10 90.156 KB 32 B 51.156 KB 64 B 9.016 KB
4-byte array (object[], int[], float[]) 1,047 502.555 KB 16 B 128.008 KB 48 B 491 B
8-byte array (long[], double[]) 205 11.672 KB 32 B 144 B 32 B 58 B
non-Java object 2 504 B 24 B 480 B 480 B 252 B
Allocation count per size
1.00
0.75
0.50
0.25
0.00
Count
Size
Name
mi_4lte-53e483a9 Online 6.0.1
com.example.baidu.sdktest 29625 8600 / 8...
LogCat
Console
DDMS
[2016-05-26 14:37:43 - DeviceMonitor] Connection attempts: 9
[2016-05-26 14:37:44 - DeviceMonitor] Connection attempts: 10
[2016-05-26 14:37:45 - DeviceMonitor] Connection attempts: 11
[2016-05-26 14:37:45 - adb] adb I 35790 504752 usb_osx.cpp:259] Found vid=2717 pid=0368 serial=53e483a9
164M of 548M

格式转换

4. hprof-conv baidu.sdktest.hprof sdktest.hprof

baidu.sdktest.hprof 是Dump HPROF file 生成的Dalvik格式文件

转成J2SE格式 sdktest.hprof 后才能由MAT打开

hprof-conv 工具在Android/sdk/platform-tools目录下

```
MacBook-Pro:platform-tools baidu$ pwd
/Users/baidu/Library/Android/sdk/platform-tools
MacBook-Pro:platform-tools baidu$ ls
NOTICE.txt              fastboot                source.properties
adb                     help                    sqlite3
api                     hprof-conv              systrace
dmtracedump             lib64
etc1tool                package.xml
MacBook-Pro:platform-tools baidu$ hprof-conv ~/Downloads/com.example.baidu.sdkte
st.hprof  ~/Downloads/sdktest.hprof
MacBook-Pro:platform-tools baidu$
```

5. 使用MAT分析转换后的sdktest.hprof文件

Baidu 百度

Eclipse Memory Analyzer

H

Property

Resource

General infor

Format

JVM versic

Time

Date

Identifier s

File path

File length

Statistic infor

Heap

Number of

Number of

Number of

Number of

H

Recently Used File

/Users/baidu

/Users/baidu

/Users/baidu

/Users/baidu

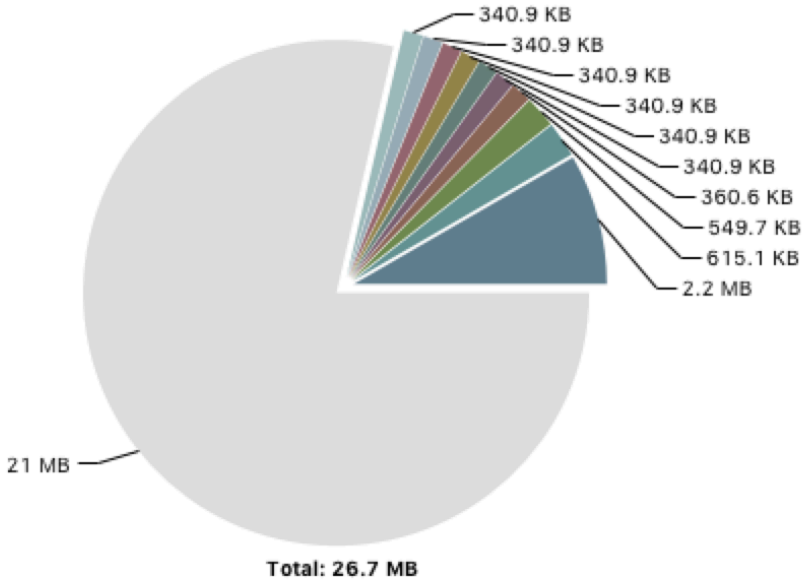
sdktest.hprof

Overview

Details

Size: **26.7 MB** Classes: **4.4k** Objects: **157.9k** Class Loader: **4** Unreachable Objects Histogram

Biggest Objects by Retained Size



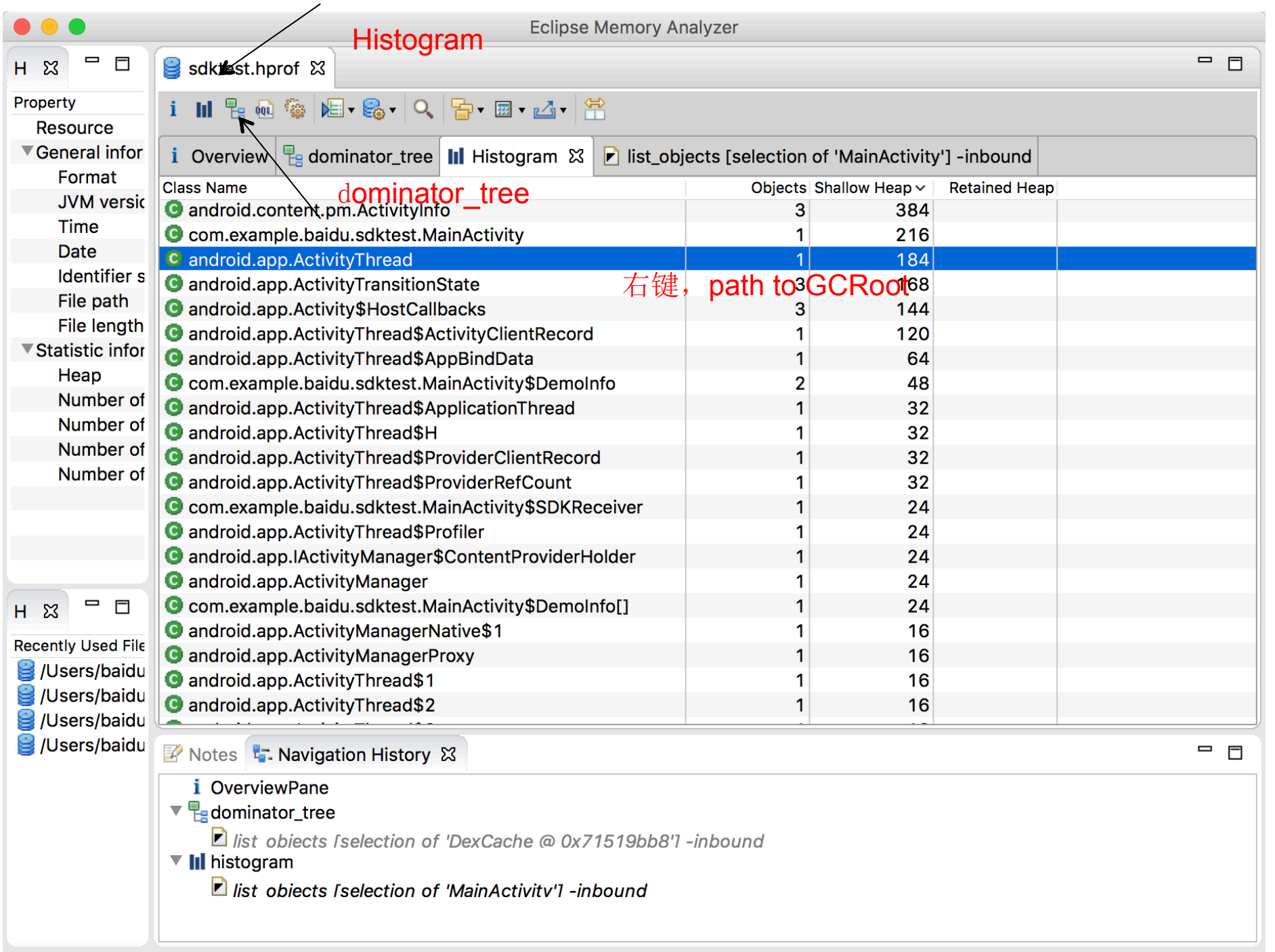
Remainder

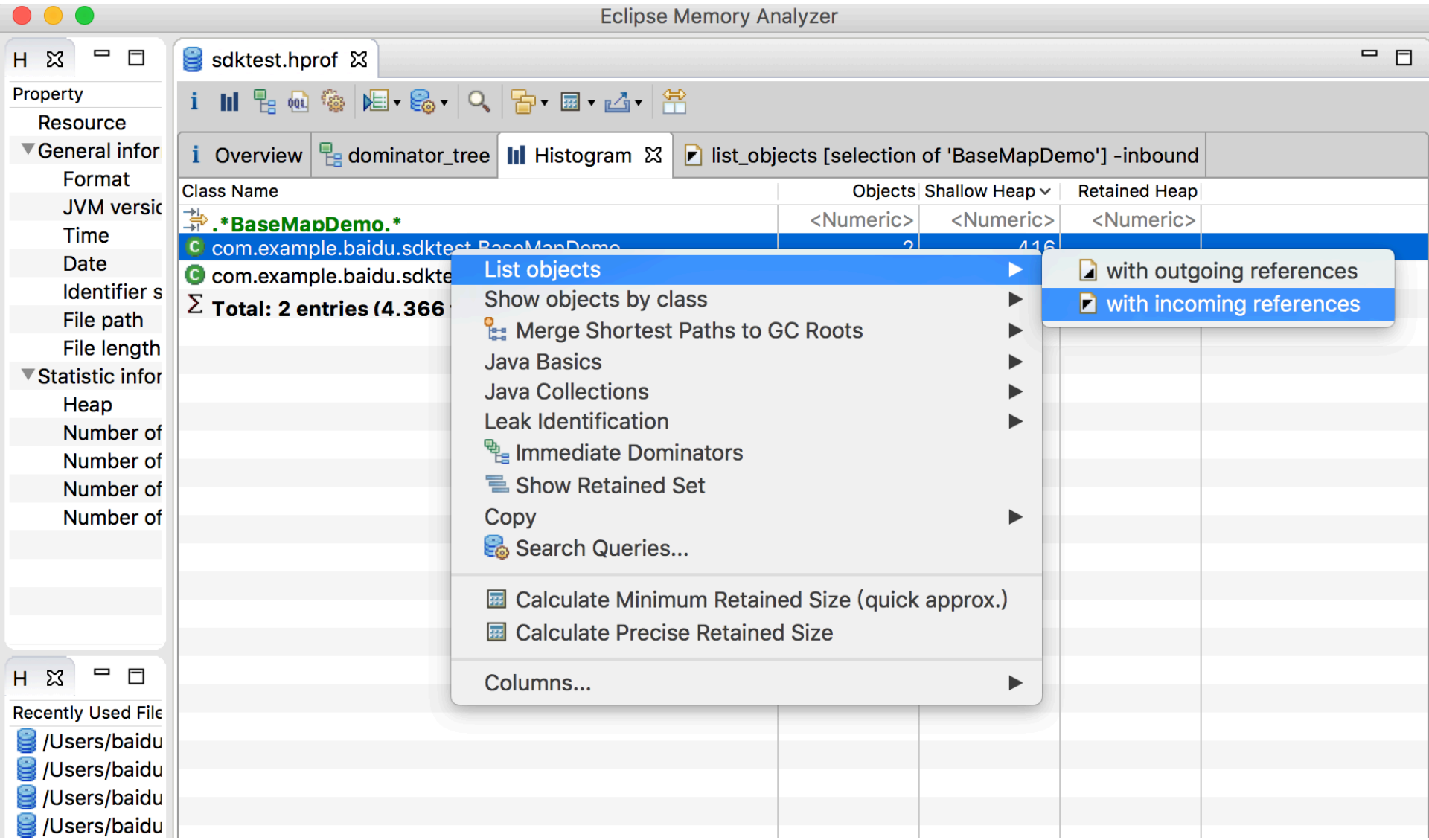
Notes

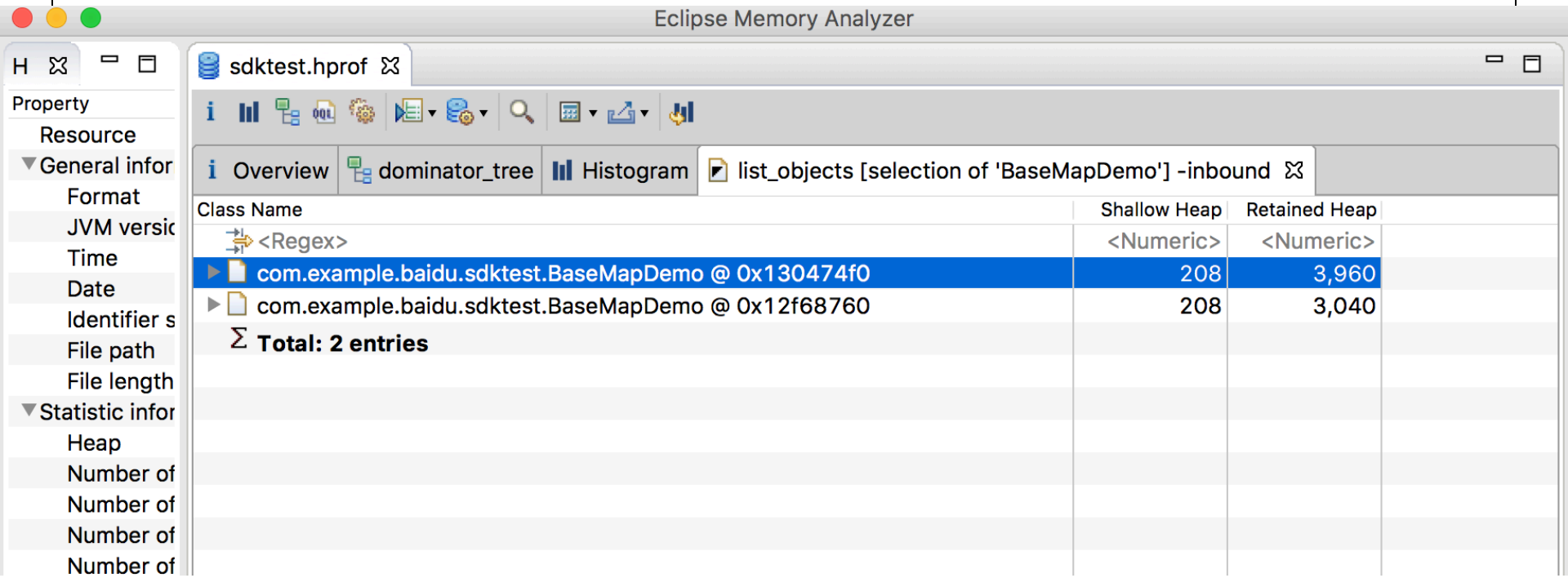
Navigation History

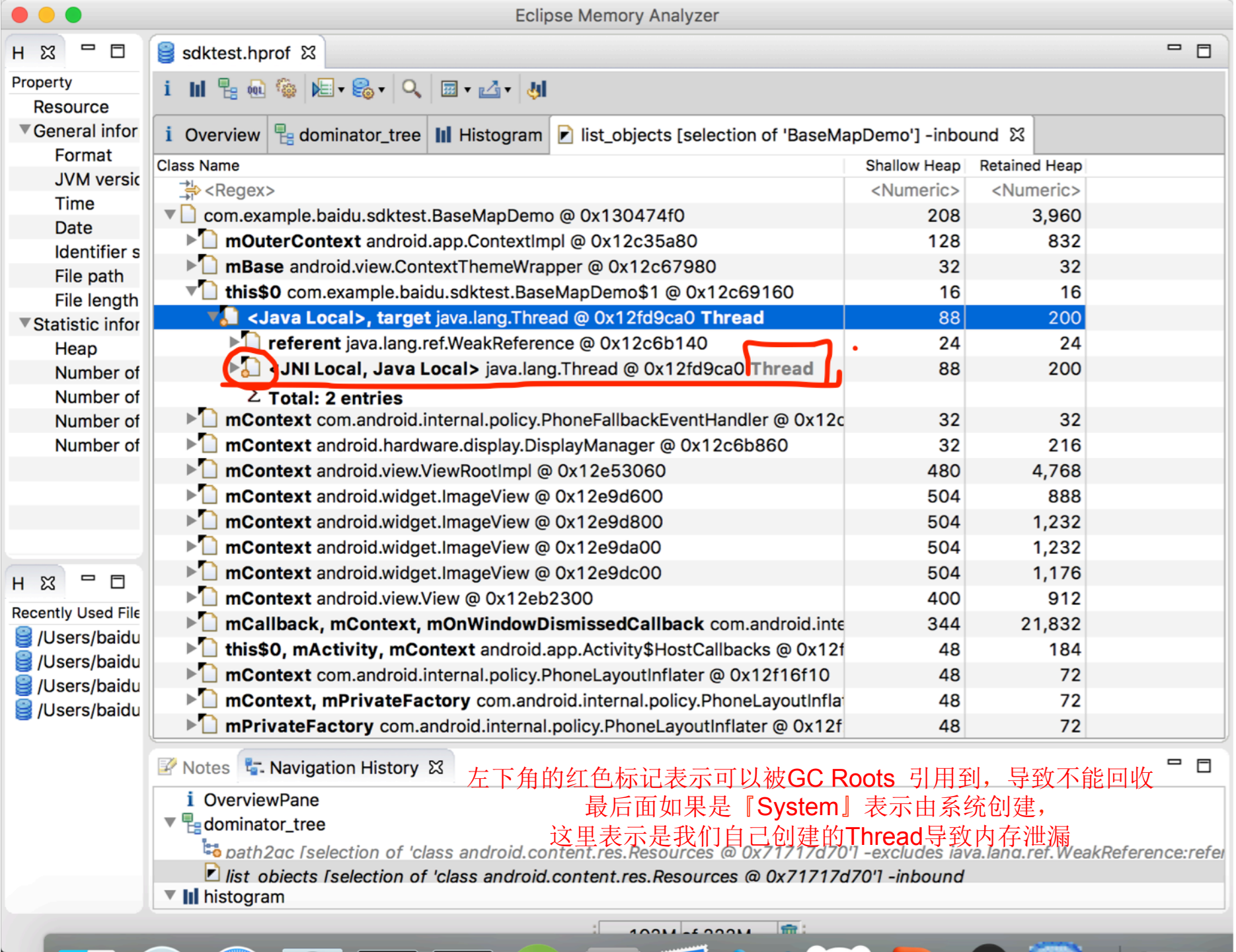
OverviewPane

105M of 218M


Eclipse Memory Analyzer
Histogram
sdktest.hprof
dominator_tree
Overview
dominator_tree
Histogram
list_objects [selection of 'MainActivity'] -inbound
右键，path to GCRoot
Class Name
Objects
Shallow Heap
Retained Heap
android.content.pm.ActivityInfo 3 384
com.example.baidu.sdktest.MainActivity 1 216
android.app.ActivityThread 1 184
android.app.ActivityTransitionState 3 168
android.app.Activity$HostCallbacks 3 144
android.app.ActivityThread$ActivityClientRecord 1 120
android.app.ActivityThread$AppBindData 1 64
com.example.baidu.sdktest.MainActivity$DemoInfo 2 48
android.app.ActivityThread$ApplicationThread 1 32
android.app.ActivityThread$H 1 32
android.app.ActivityThread$ProviderClientRecord 1 32
android.app.ActivityThread$ProviderRefCount 1 32
com.example.baidu.sdktest.MainActivity$SDKReceiver 1 24
android.app.ActivityThread$Profiler 1 24
android.app.IActivityManager$ContentProviderHolder 1 24
android.app.ActivityManager 1 24
com.example.baidu.sdktest.MainActivity$DemoInfo[] 1 24
android.app.ActivityManagerNative$1 1 16
android.app.ActivityManagerProxy 1 16
android.app.ActivityThread$1 1 16
android.app.ActivityThread$2 1 16
Property
Resource
General infor
Format
JVM versic
Time
Date
Identifier s
File path
File length
Statistic infor
Heap
Number of
Number of
Number of
Number of
Recently Used File
/Users/baidu
/Users/baidu
/Users/baidu
/Users/baidu
Notes
Navigation History
OverviewPane
dominator_tree
list objects [selection of 'DexCache @ 0x71519bb8'] -inbound
histogram
list objects [selection of 'MainActivity'] -inbound


Eclipse Memory Analyzer
sdktest.hprof
Property
Resource
General infor
Format
JVM versic
Time
Date
Identifier s
File path
File length
Statistic infor
Heap
Number of
Number of
Number of
Number of
Recently Used File
/Users/baidu
/Users/baidu
/Users/baidu
/Users/baidu
Overview
dominator_tree
Histogram
list_objects [selection of 'BaseMapDemo'] -inbound
Class Name
Objects
Shallow Heap
Retained Heap
.*BaseMapDemo.*
<Numeric>
<Numeric>
<Numeric>
com.example.baidu.sdktest.BaseMapDemo
com.example.baidu.sdkte
Total: 2 entries (4,366
List objects
Show objects by class
Merge Shortest Paths to GC Roots
Java Basics
Java Collections
Leak Identification
Immediate Dominators
Show Retained Set
Copy
Search Queries...
Calculate Minimum Retained Size (quick approx.)
Calculate Precise Retained Size
Columns...
with outgoing references
with incoming references


Eclipse Memory Analyzer
sdktest.hprof
Property
Resource
General infor
Format
JVM versic
Time
Date
Identifier s
File path
File length
Statistic infor
Heap
Number of
Number of
Number of
Number of
Overview
dominator_tree
Histogram
list_objects [selection of 'BaseMapDemo'] -inbound
Class Name
Shallow Heap
Retained Heap
<Regex>
<Numeric>
<Numeric>
com.example.baidu.sdktest.BaseMapDemo @ 0x130474f0
208
3,960
com.example.baidu.sdktest.BaseMapDemo @ 0x12f68760
208
3,040
Total: 2 entries


Eclipse Memory Analyzer
sdktest.hprof
Overview
dominator_tree
Histogram
list_objects [selection of 'BaseMapDemo'] -inbound
Class Name
Shallow Heap
Retained Heap
<Regex>
<Numeric>
<Numeric>
com.example.baidu.sdktest.BaseMapDemo @ 0x130474f0 208 3,960
mOuterContext android.app.ContextImpl @ 0x12c35a80 128 832
mBase android.view.ContextThemeWrapper @ 0x12c67980 32 32
this$0 com.example.baidu.sdktest.BaseMapDemo$1 @ 0x12c69160 16 16
<Java Local>, target java.lang.Thread @ 0x12fd9ca0 Thread 88 200
referent java.lang.ref.WeakReference @ 0x12c6b140 24 24
<JNI Local, Java Local> java.lang.Thread @ 0x12fd9ca0 Thread 88 200
Total: 2 entries
mContext com.android.internal.policy.PhoneFallbackEventHandler @ 0x12c 32 32
mContext android.hardware.display.DisplayManager @ 0x12c6b860 32 216
mContext android.view.ViewRootImpl @ 0x12e53060 480 4,768
mContext android.widget.ImageView @ 0x12e9d600 504 888
mContext android.widget.ImageView @ 0x12e9d800 504 1,232
mContext android.widget.ImageView @ 0x12e9da00 504 1,232
mContext android.widget.ImageView @ 0x12e9dc00 504 1,176
mContext android.view.View @ 0x12eb2300 400 912
mCallback, mContext, mOnWindowDismissedCallback com.android.inte 344 21,832
this$0, mActivity, mContext android.app.Activity$HostCallbacks @ 0x12f 48 184
mContext com.android.internal.policy.PhoneLayoutInflater @ 0x12f16f10 48 72
mContext, mPrivateFactory com.android.internal.policy.PhoneLayoutInfla 48 72
mPrivateFactory com.android.internal.policy.PhoneLayoutInflater @ 0x12f 48 72
Property
Resource
General infor
Format
JVM versic
Time
Date
Identifier s
File path
File length
Statistic infor
Heap
Number of
Number of
Number of
Number of
Recently Used File
/Users/baidu
/Users/baidu
/Users/baidu
/Users/baidu
Notes
Navigation History
OverviewPane
dominator_tree
path2gc [selection of 'class android.content.res.Resources @ 0x71717d70'] -excludes java.lang.ref.WeakReference:refer
list_objects [selection of 'class android.content.res.Resources @ 0x71717d70'] -inbound
histogram
左下角的红色标记表示可以被GC Roots 引用到，导致不能回收
最后面如果是『System』表示由系统创建，
这里表示是我们自己创建的Thread导致内存泄漏

总结

◆1.尽量不要让生命周期可能长于Activity的对象持有其引用

◆2.资源不使用时，及时释放

◆3.使用工具DDMS+MAT帮助分析内存泄漏

Baidu 百度

That’s all

Thanks!

Baidu 百度