

Android 内存泄漏问题的排查

<http://liubin.nanshapo.com>

首先先检查下你有没有犯这些错误（内存泄露的主要原因）：

生命周期过长的对象（static），尤其是集合对象（List/Map等）作为cache等使用，如果没有将某个对象主动的从中清除的话，这个集合就会占用越来越多的内存，可以用WeakReference，如WeakHashMap，使得它持有的对象不增加对象的引用数。

Scope定义不对，方法的局部变量定义成类的变量，类的静态变量等。尽量使得变量作用域别太大。

异常时没有加finally{}来释放某些资源，比如Cursor。

Listener没有显式remove；内部类持有外部对象的隐式引用，不论是什么，如果有add方法，一定要想想是否需要remove方法。

图像对象是否在不用的时候释放了（((BitmapDrawable) d).getBitmap().recycle();）

以下是正文

1. 获取 hprof 信息

1.1. 使用脚本 get_hprof.sh

```
./get_hprof.sh  
-- 脚本内容见附录
```

运行此脚本之前，需要根据你的实际情况修改脚本，设置一下几个变量：

```
# where is you sdk
```

```
SDKPATH=.
```

这个是需要制定 sdk/tools 目录的位置，需要执行其下的 adb 和 hprof-conv 命令

```
# where will you want to save the log file
```

```
TARGET_LOG_DIR=./hproflog
```

log 的保存位置

```
# which process will you want to generate log
```

```
# etc,mms or com.android.mms for the full process name
```

```
# we will grep the NAME field of `ps` command
```

```
# to identify the process id
```

```
PKG=mms
```

需要取得的进程名或者一部分，比如 mms 可以取得 com.android.mms 的进程信息

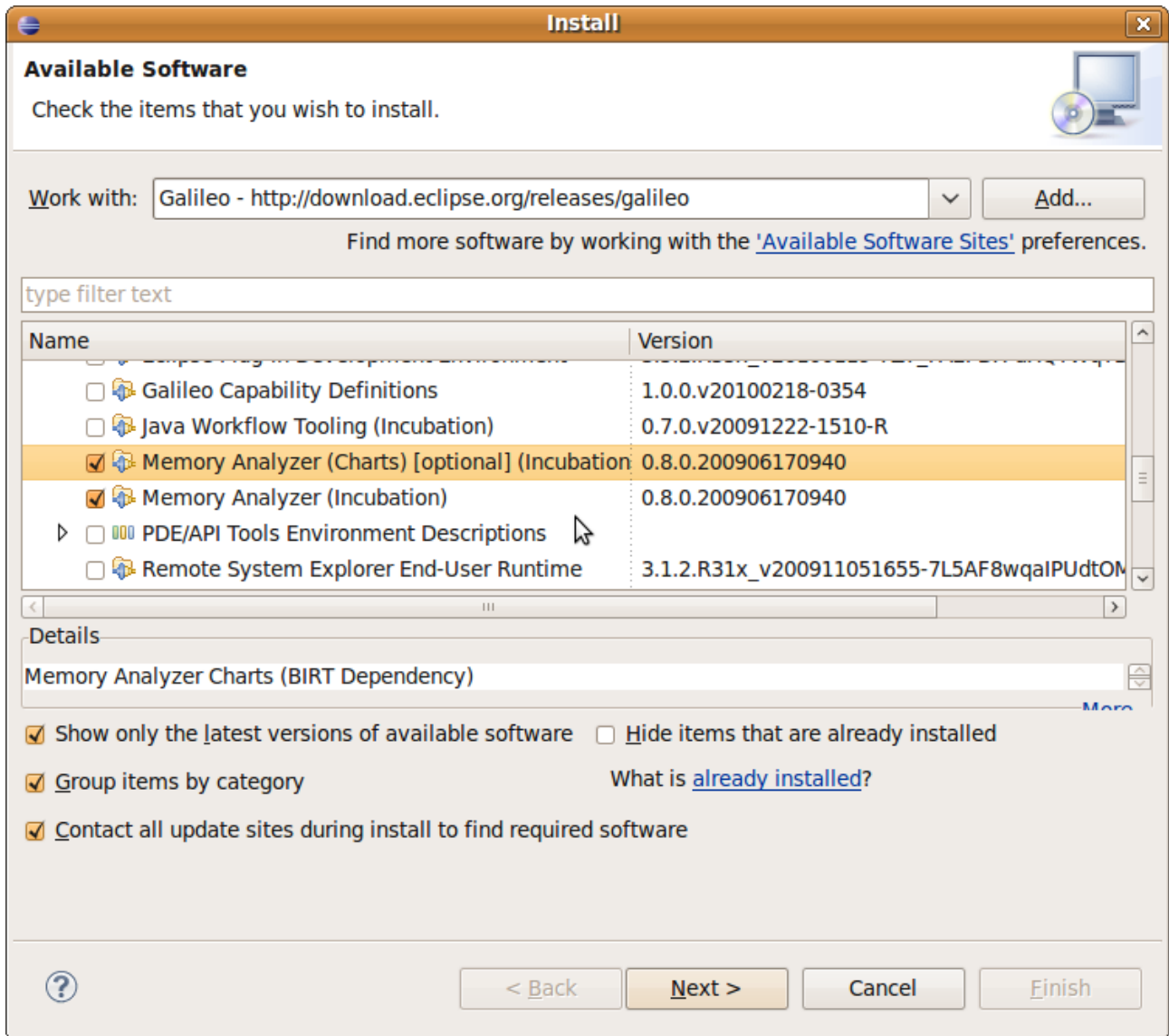
```
# eclipse project path to show the hprof report
```

```
ECLIPSE_PROJECT_PATH=/home/liubin/workspace_old/hprof/
```

取得.hprof 文件会被自动拷贝到 eclipse 工程下，用 mat 进行分析

1.2. MAT 的安装

eclipse 3.5的话，直接在 install new software 菜单里安装：



其它版本的安装请参考 eclipse 的官方网站。

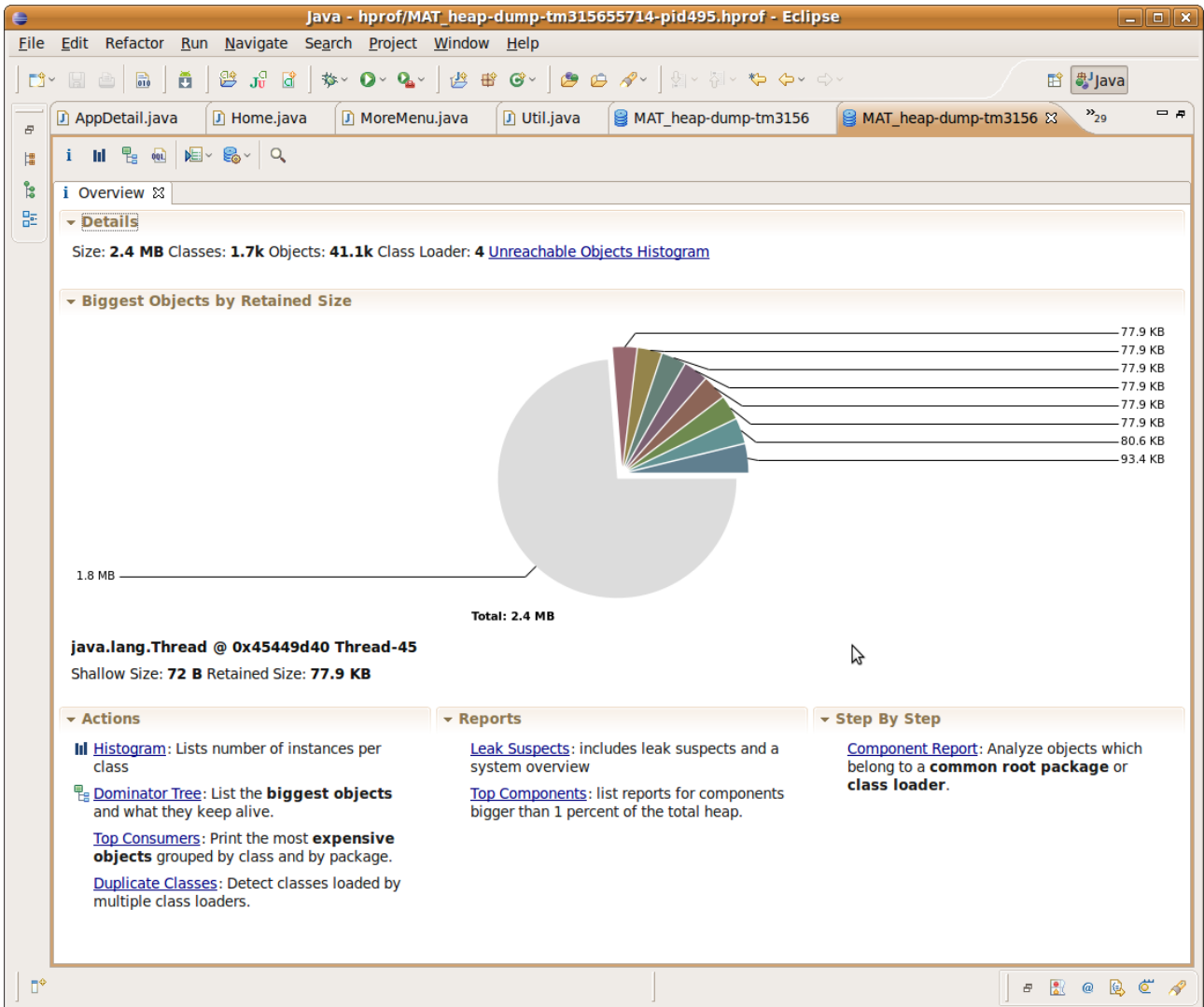
2. 使用 MAT

如果安装成功 MAT，就可以查看 hprof 报告了，在项目里双击生成的 .hprof 文件，即可查看。双击之后会弹出一个选择框，询问我们会干什么，选在第一个生成可疑报告就行了（不选也可以）：



当然，如果不想每次都显示这个窗口，可以去掉界面中唯一一个可以打勾的地方。

如果在上面的界面里选择取消，则先显示主界面如下：

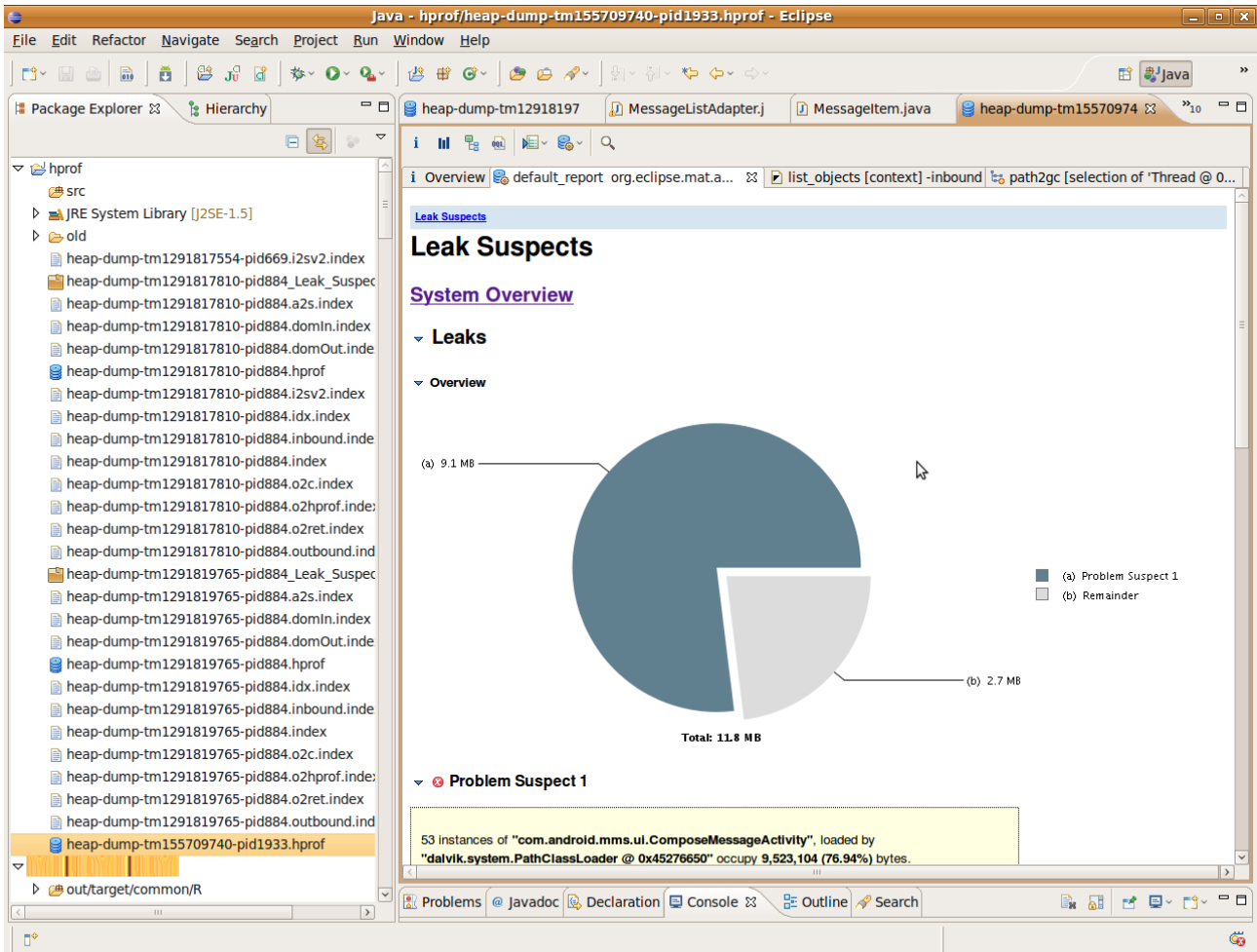


Overview 里面的下方，有 3 栏菜单，第一个柱状图用来显示各个对象的情况，DominatorTree 则显示占用 heap 比较多的对象，类似的，Top Consumers 等，看说明都能知道是干什么的。

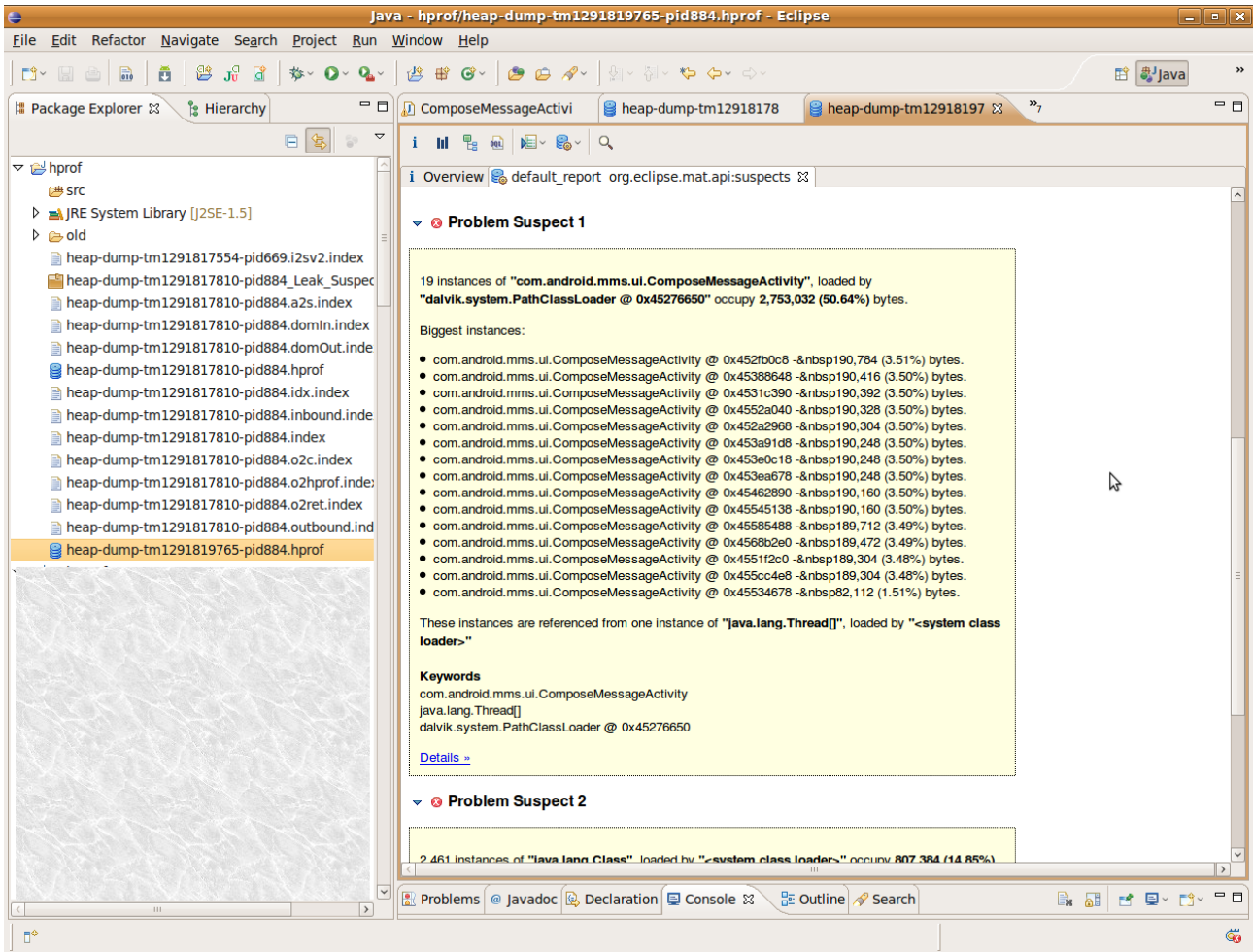
这里只说两个方面的，一个是 Reports 栏的 Leak Suspects,生成可以报告，和 DominatorTree，查找比较占用内存的对象。

2.1. Leak Suspects

单击 Overview 页面的 [LeakSuspects] 链接，即可生产可以报告，如图所示。对于可疑的对象，会罗列出来，比如 problem suspect1，problem suspect2 等，并且以饼状图显示出来。



并且，会在下面接着列出来主要的（值得怀疑的）对象：



点击了"Details"链接之后，除了在上一页看到的描述外，还有 Shortest Paths To the Accumulation Point 和 Accumulated Objects 部分，这里说明了从 GC root 到聚集点的最短路径，以及完整的 reference chain。每个带链接的东西都是可以点的。

接着往下看，来到了 Accumulated Objects by Class 区域，顾名思义，这里能找到被聚集的对象实例的类名

2.2. Dominator tree 视图

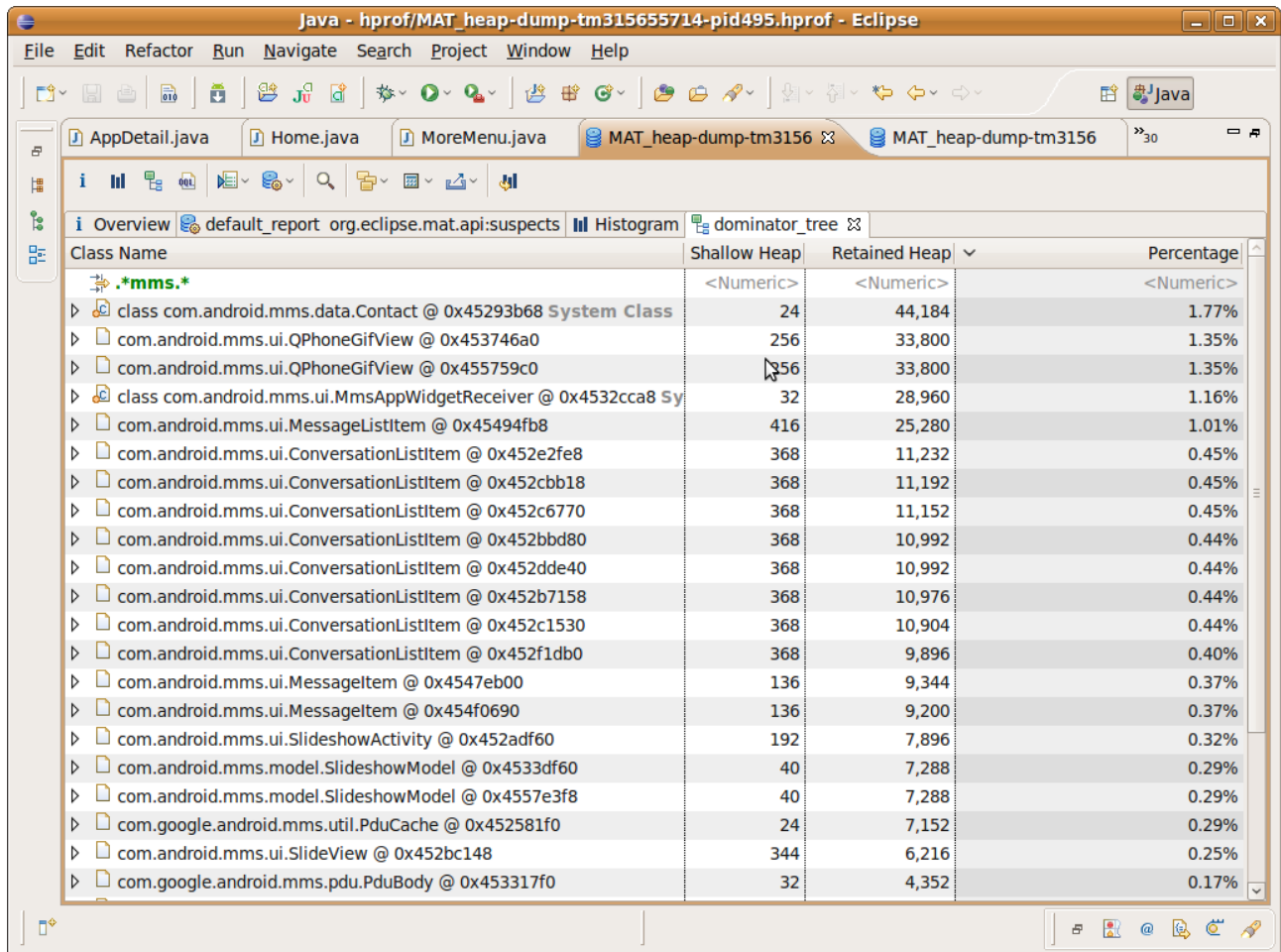
打开这个视图，会看到顶层的对象及占用 heap 的大小和百分比。单击对象前的三角形，可以查看对象间的（持有）关系。（注意，不一定是引用关系）

按百分比或者 Retained Heap 倒序排序，即可发现有问题的对象

shallow heap 是一个对象消耗掉的内存

retained heap 是这个对象如果被回收的话，总共会释放多少内存

比如对象 a 保有 b 和 c 两个对象，如果 a 被回收，则 b 和 c 的内存也会连带被回收。



Class Name	Shallow Heap	Retained Heap	Percentage
.*mms.*	<Numeric>	<Numeric>	<Numeric>
class com.android.mms.data.Contact @ 0x45293b68 System Class	24	44,184	1.77%
com.android.mms.ui.QPhoneGifView @ 0x453746a0	256	33,800	1.35%
com.android.mms.ui.QPhoneGifView @ 0x455759c0	256	33,800	1.35%
class com.android.mms.ui.MmsAppWidgetReceiver @ 0x4532cca8 Sy	32	28,960	1.16%
com.android.mms.ui.MessageListItem @ 0x45494fb8	416	25,280	1.01%
com.android.mms.ui.ConversationListItem @ 0x452e2fe8	368	11,232	0.45%
com.android.mms.ui.ConversationListItem @ 0x452cbb18	368	11,192	0.45%
com.android.mms.ui.ConversationListItem @ 0x452c6770	368	11,152	0.45%
com.android.mms.ui.ConversationListItem @ 0x452bbd80	368	10,992	0.44%
com.android.mms.ui.ConversationListItem @ 0x452dde40	368	10,992	0.44%
com.android.mms.ui.ConversationListItem @ 0x452b7158	368	10,976	0.44%
com.android.mms.ui.ConversationListItem @ 0x452c1530	368	10,904	0.44%
com.android.mms.ui.ConversationListItem @ 0x452f1db0	368	9,896	0.40%
com.android.mms.ui.MessageItem @ 0x4547eb00	136	9,344	0.37%
com.android.mms.ui.MessageItem @ 0x454f0690	136	9,200	0.37%
com.android.mms.ui.SlideshowActivity @ 0x452adf60	192	7,896	0.32%
com.android.mms.model.SlideshowModel @ 0x4533df60	40	7,288	0.29%
com.android.mms.model.SlideshowModel @ 0x4557e3f8	40	7,288	0.29%
com.google.android.mms.util.PduCache @ 0x452581f0	24	7,152	0.29%
com.android.mms.ui.SlideView @ 0x452bc148	344	6,216	0.25%
com.google.android.mms.pdu.PduBody @ 0x453317f0	32	4,352	0.17%

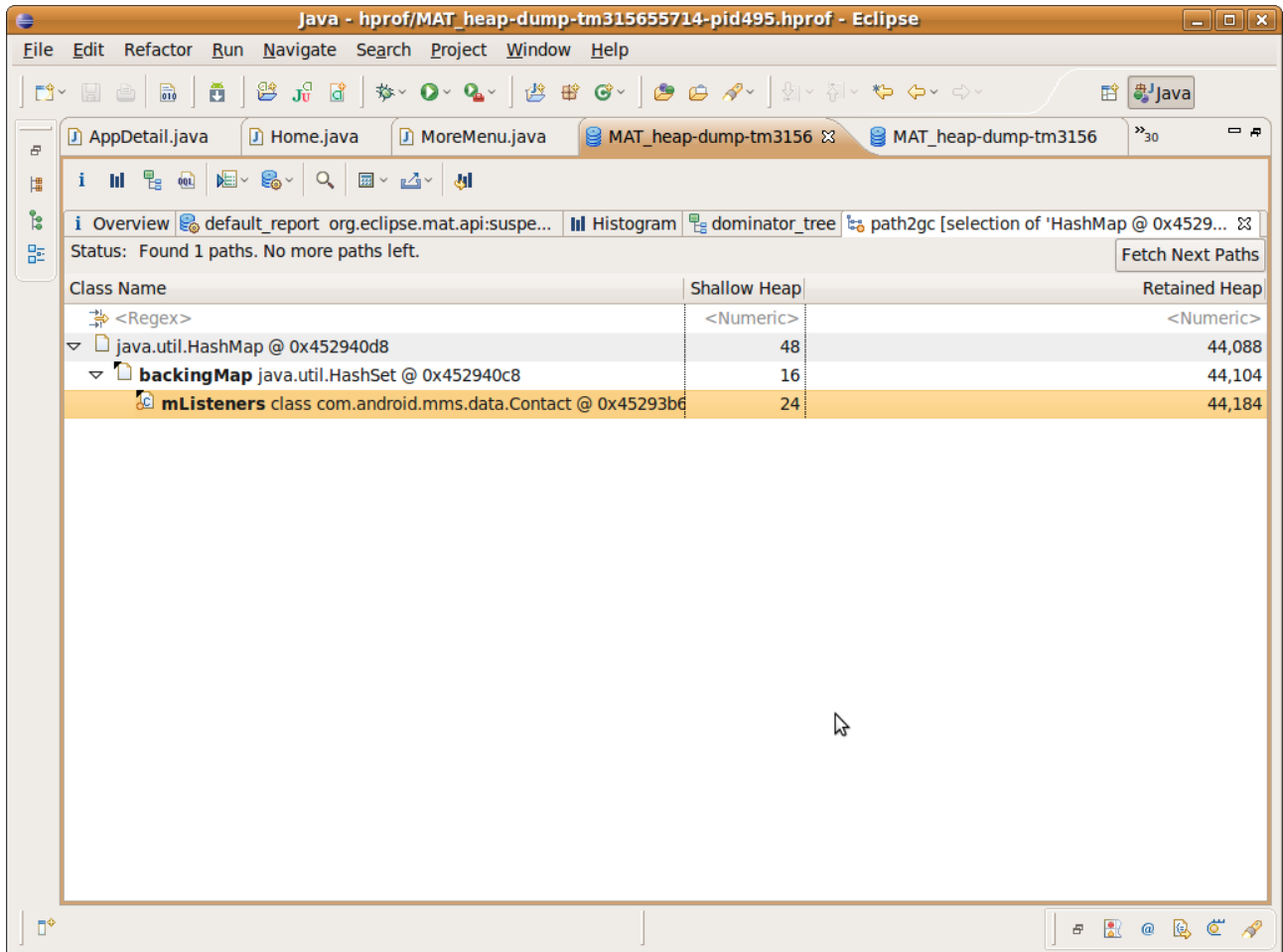
注意：ClassName 栏的第一行可以输入 filter，比如 mms 等，其它界面也可以进行对结果的过滤。

展开某一对象（如上图的第一条），即可查看其持有 heap 详细：

注意，某一对象的 Retained Heap 等数值是其子节点的和，孙子不能重复计算。

Class Name	Shallow Heap	Retained Heap	Percentage
<Numeric>	<Numeric>	<Numeric>	<Numeric>
class com.android.mms.data.Contact @ 0x45293b68 System Class	24	44,184	1.77%
java.util.HashSet @ 0x452940c8	16	44,104	1.76%
java.util.HashMap @ 0x452940d8	48	44,088	1.76%
java.util.HashMap\$HashMapEntry[16] @ 0x45225268	80	44,024	1.76%
java.util.HashMap\$HashMapEntry @ 0x452bd780	24	21,832	0.87%
com.android.mms.ui.ConversationListItem @ 0x45223bc0	368	11,016	0.44%
java.util.HashMap\$HashMapEntry @ 0x452252c0	24	10,792	0.43%
Σ Total: 2 entries			
java.util.HashMap\$HashMapEntry @ 0x452eb270	24	11,040	0.44%
com.android.mms.ui.ConversationListItem @ 0x4528f3b0	368	11,016	0.44%
java.util.HashMap\$HashMapEntry @ 0x452f5760	24	10,904	0.44%
java.util.HashMap\$HashMapEntry @ 0x452f0698	24	10,880	0.43%
java.util.HashMap\$HashMapEntry @ 0x452ca120	24	72	0.00%
java.util.HashMap\$HashMapEntry @ 0x452e18d0	24	48	0.00%
java.util.HashMap\$HashMapEntry @ 0x452c5330	24	24	0.00%
java.util.HashMap\$HashMapEntry @ 0x452e69d0	24	24	0.00%
Σ Total: 7 entries			
java.util.HashMap\$EntrySet @ 0x452304e0	16	16	0.00%
Σ Total: 2 entries			
com.android.mms.data.Contact\$1 @ 0x45293fd0	24	56	0.00%
android.os.Handler @ 0x45294040	24	24	0.00%

想知道是谁引用了占有大量 heap 空间的对象而导致其不能被释放的原因，可以在选中的对象上点击右键，选择“Path to GC Roots” -> “exclude weak/ soft references”，即可显示引用关系，见下图所示。从图中，可以看到，问题应该出现在 mListeners 这个变量上了。



附录 1

推荐几个网址：

<http://www.eclipse.org/mat/>

<http://www.sdn.sap.com/ir,j/scn/weblogs?blog=/pub/wlg/7680>

<http://blog.csdn.net/xtyyumi301/archive/2008/10/04/3015493.aspx>

附录 2

```
#!/bin/sh
```

```
# author liubin
```

```
# get_hprof.sh 2010/12/17
```

```
#####  
#####  
##### you may change this area #####  
#####  
#####
```

```
# where is you sdk
```

```
SDKPATH=.
```

```
# where will you want to save the log file
```

```
TARGET_LOG_DIR=./hproflog
```

```
# which process will you want to generate log
```

```
# etc,mms or com.android.mms for the full process name
```

```
# we will grep the NAME field of `ps` command
```

```
# to identify the process id
```

```
PKG=mms
```

```
# eclipse project path to show the hprof report
```

```
ECLIPSE_PROJECT_PATH=/home/liubin/workspace_old/hprof/
```

```
#####  
#####  
##### you may NOT change this area #####  
#####  
#####
```

```
# adb path
```

```
ADB=$SDKPATH/adb
```

```
# hprof-conv path
```

```
HPROFCONV=$SDKPATH/hprof-conv
```

```
#echo $ADB
```

```
#exit

# check if the log directory exist
if [ -d $TARGET_LOG_DIR ]; then
    echo "using Log directory: $TARGET_LOG_DIR"
else
    echo "create not exists directory $TARGET_LOG_DIR"
    mkdir $TARGET_LOG_DIR
    if [ "$?" != 0 ]; then
        echo "Create Log directory ERROR !!!!!"
        exit 1
    fi
    echo "using new log directory: $TARGET_LOG_DIR"
fi

# make /data/misc writable
$ADB shell chmod 777 /data/misc

# fetch the pid of $PKG
pid=`$ADB shell ps | grep $PKG | awk '{print $2;}'`
echo "$PKG pid is $pid"

# send -10 signal to the $PKG process to generate a log file
$ADB shell kill -10 $pid

# select 3 sencond to wait for the complete of hprof file creating.
# if not wait ,we may get the previous log
sleep 3

# get the latest log file name
pfile=`$ADB shell ls -1 /data/misc | grep hprof | grep $pid | awk '{print $7;}'`
| sort -r | head -n 1 | sed -e 's/[\n\r]//g'`

#pfile="/data/misc/$pfile"
echo "new hprof file is $pfile"

# pull to target directory
$ADB pull "/data/misc/$pfile" $TARGET_LOG_DIR

if [ "$?" != 0 ]; then
    echo "adb pull error!"
else
```

```
    echo "pull log from avd ok"
fi

# convert to mat format
$HPROFCONV $TARGET_LOG_DIR/$pfile $TARGET_LOG_DIR"/MAT_"$pfile

if [ "$S?" != 0 ]; then
    echo "convert to mat format error!"
else
    echo "convert to mat format ok"
fi

# copy to eclipse project
cp $TARGET_LOG_DIR"/MAT_"$pfile $ECLIPSE_PROJECT_PATH/

if [ "$S?" != 0 ]; then
    echo "copy to eclipse project error !"
else
    echo "copy to eclipse project ok"
fi
```