内存泄露从入门到精通三部曲之排查方法篇

2015.11.11 腾讯Bugly 微信分享

1 最原始的内存泄露测试

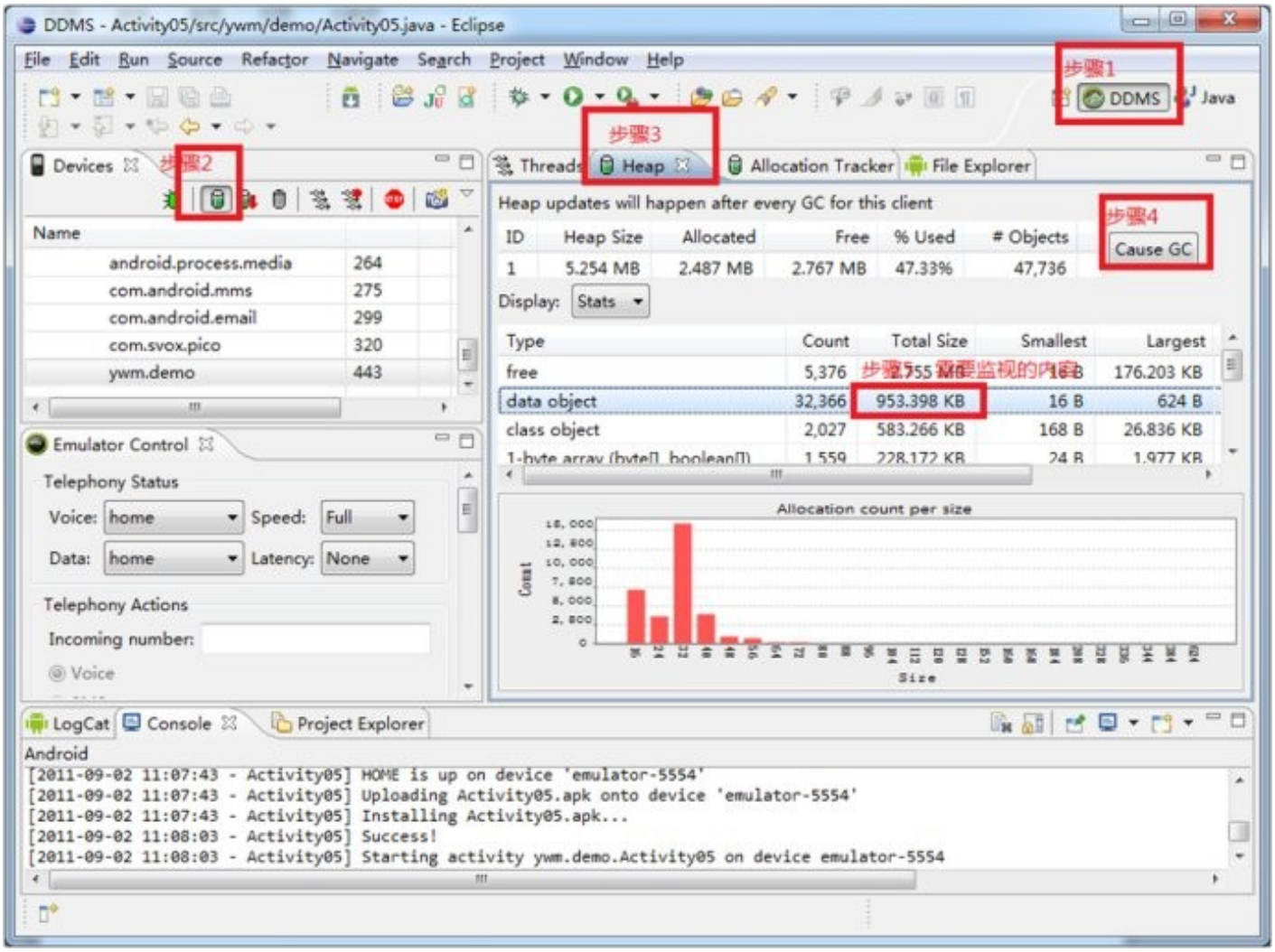
重复多次操作关键的可疑的路径，从内存监控工具中观察内存曲线，是否存在不断上升的趋势且不会在程序返回时明显回落。

这种方式可以发现最基本，也是最明显的内存泄露问题，对用户价值最大，操作难度小，性价比极高。

2 MAT内存分析工具

2.1 MAT分析heap的总内存占用大小来初步判断是否存在泄露

> 在Devices 中，点击要监控的程序。
>
> 点击Devices视图界面中最上方一排图标中的“Update Heap”
>
> 点击Heap视图
>
> 点击Heap视图中的“Cause GC”按钮
>
> 到此为止需检测的进程就可以被监视。



Heap视图中部有一个Type叫做data object，即数据对象，也就是我们的程序中大量存在的类类型的对象。在data object一行中有一列是“Total Size”，其值就是当前进程中所有Java数据对象的内存总量，一般情况下，这个值的大小决定了是否会有内存泄漏。可以这样判断：

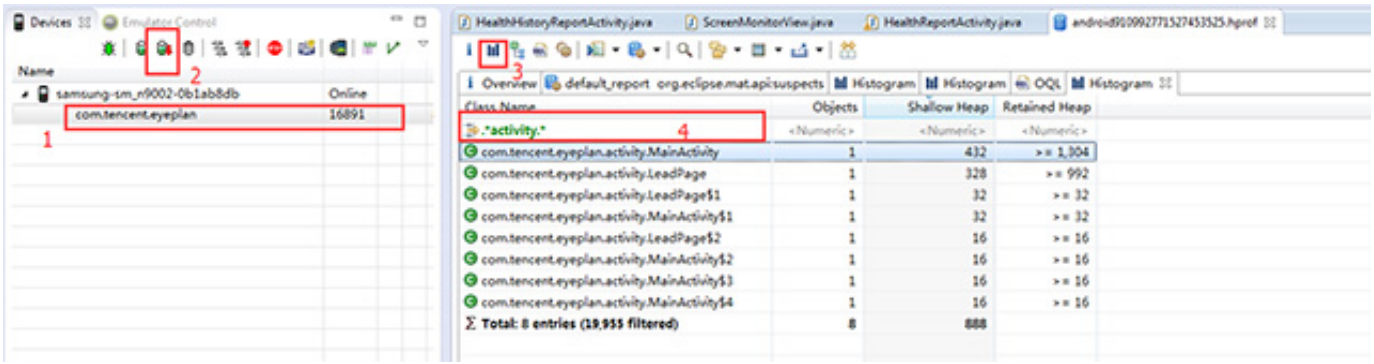
进入某应用，不断的操作该应用，同时注意观察data object的Total Size值，正常情况下Total Size值都会稳定在一个有限的范围内，也就是说由于程序中的的代码良好，没有造成对象不被垃圾回收的情况。

所以说虽然我们不断的操作会不断的生成很多对象，而在虚拟机不断的进行GC的过程中，这些对象都被回收了，内存占用量会会落到一个稳定的水平；反之如果代码中存在没有释放对象引用的情况，则data object的Total Size值在每次GC后不会有明显的回落。随着操作次数的增多Total Size的值会越来越大，直到到达一个上限后导致进程被杀掉。

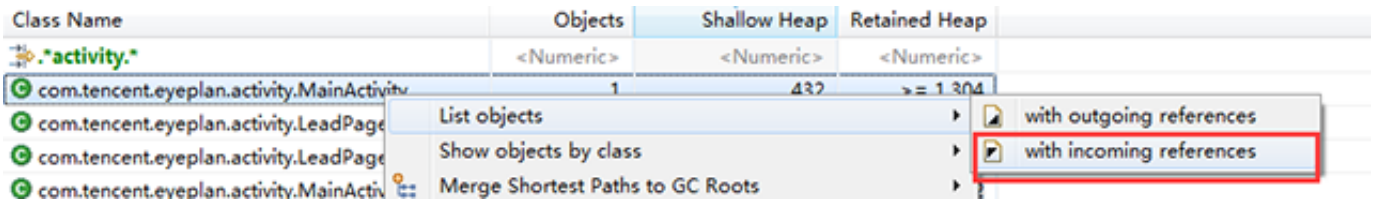
2.2 MAT分析hprof来定位内存泄露的原因所在。

这是出现内存泄露后使用MAT进行问题定位的有效手段。

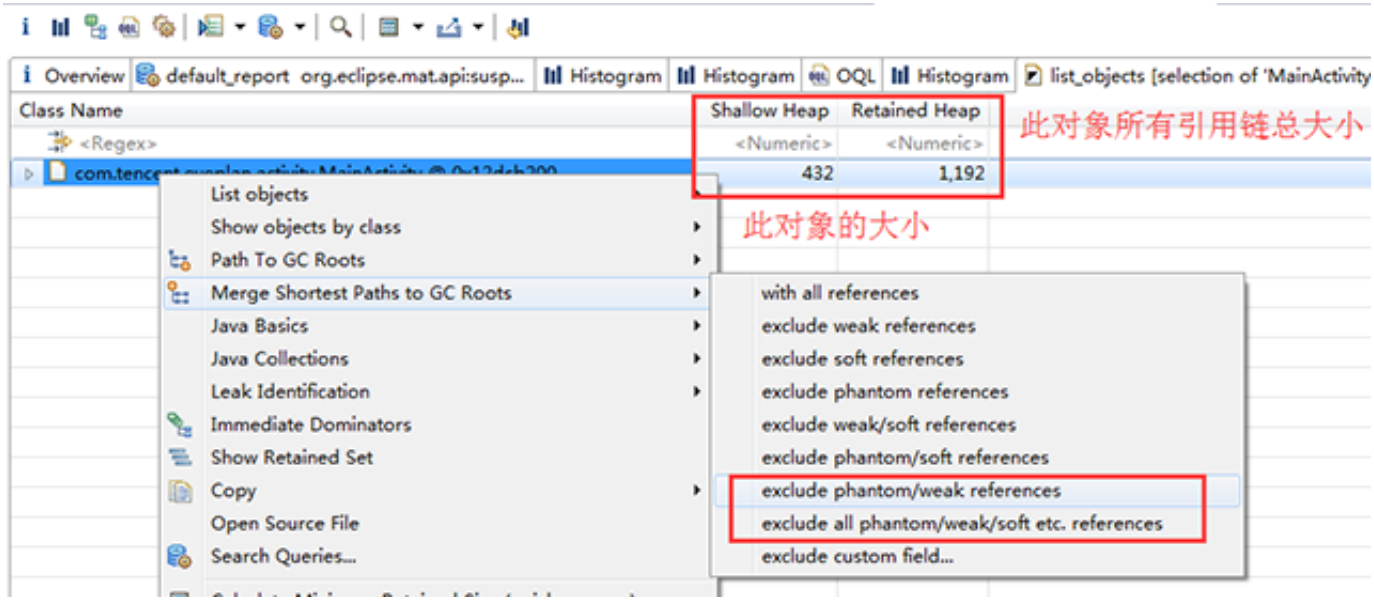
A)Dump出内存泄露当时的内存镜像hprof，分析怀疑泄露的类：



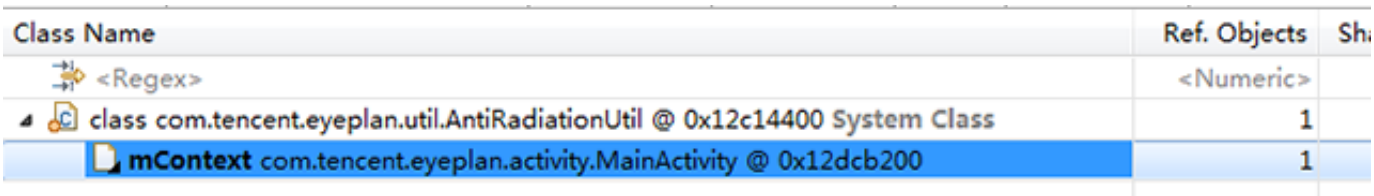
B)分析持有此类对象引用的外部对象



C)分析这些持有引用的对象的GC路径



D)逐个分析每个对象的GC路径是否正常



从这个路径可以看出是一个antiRadiationUtil工具类对象持有了MainActivity的引用导致MainActivity无法释放。此时就要进入代码分析此时antiRadiationUtil的引用持有是否合理（如果antiRadiationUtil持有了MainActivity的context导致节目退出后MainActivity无法销毁，那一般都属于内存泄露了）。

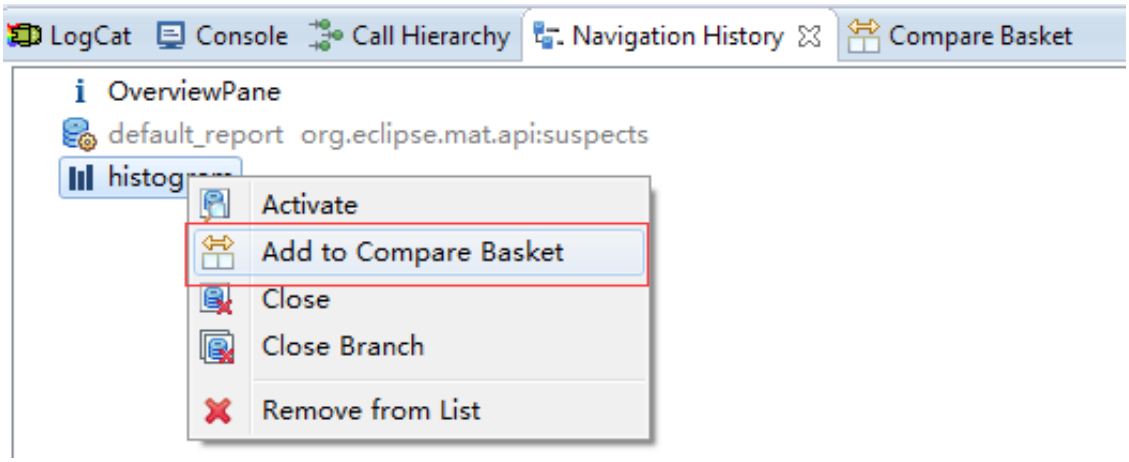
2.3 MAT对比操作前后的hprof来定位内存泄露的根因所在。

为查找内存泄漏，通常需要两个 Dump结果作对比，打开 Navigator History面板，将两个表的 Histogram 结果都添加到 Compare Basket中去

A） 第一个HPROF 文件(usingFile > Open Heap Dump).

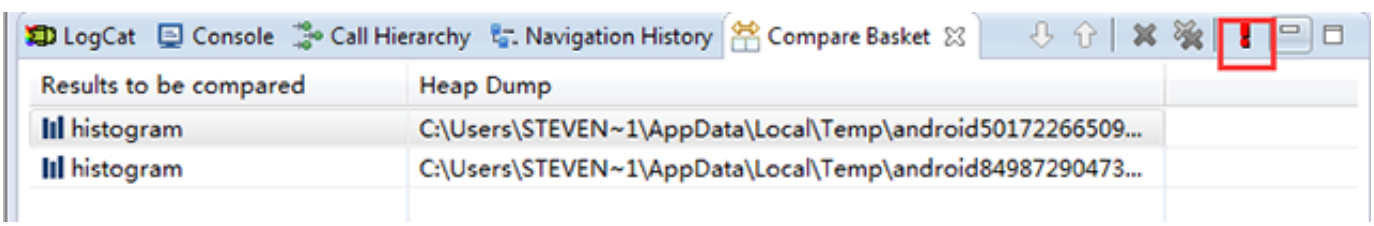
B）打开Histogram view.

C）在NavigationHistory view里 (如果看不到就从Window >show view>MAT- Navigation History), 右击histogram然后选择Add to Compare Basket .

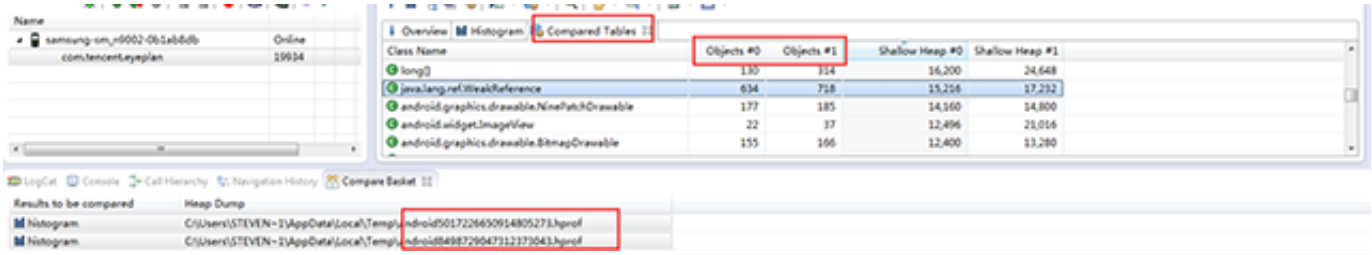


D）打开第二个HPROF 文件然后重做步骤2和3.

E）切换到Compare Basket view, 然后点击Compare the Results (视图右上角的红色”!”图标)。



F）分析对比结果



可以看出两个hprof的数据对象对比结果。

通过这种方式可以快速定位到操作前后所持有的对象增量，从而进一步定位出当前操作导致内存泄露的具体原因是泄露了什么数据对象。

注意：

如果是用 MAT Eclipse 插件获取的 Dump文件，不需要经过转换则可在MAT中打开，Adt会自动进行转换。

而手机SDk Dump 出的文件要经过转换才能被 MAT识别，Android SDK提供了这个工具 hprof-conv (位于sdk/tools下)

首先，要通过控制台进入到你的 android sdk tools 目录下执行以下命令：

./hprof-conv xxx-a.hprof xxx-b.hprof

例如 hprof-conv input.hprof out.hprof

此时才能将out.hprof放在eclipse的MAT中打开。

3 手机管家内存泄露每日监控方案

目前手机管家的内存泄露每日监控会自动运行并输出是否存在疑似泄露的报告邮件，不论泄露对象的大小。这其中涉及的核心技术主要是AspectJ，MLD自研工具（原理是虚引用）和UIAutomator。

3.1 AspectJ插桩监控代码

手机管家目前使用一个ant脚本加入MLD的监控代码，并通过AspectJ的语法实现插桩。

使用AspectJ的原因是可以灵活分离出项目源码与监控代码，通过不同的编译脚本打包出不同用途的安装测试包：如果测试包是经过Aspect插桩了MLD监控代码的话，那么运行完毕后会输出指定格式的日志文件，作为后续分析工作的数据基础。

3.2 MLD实现监控核心逻辑

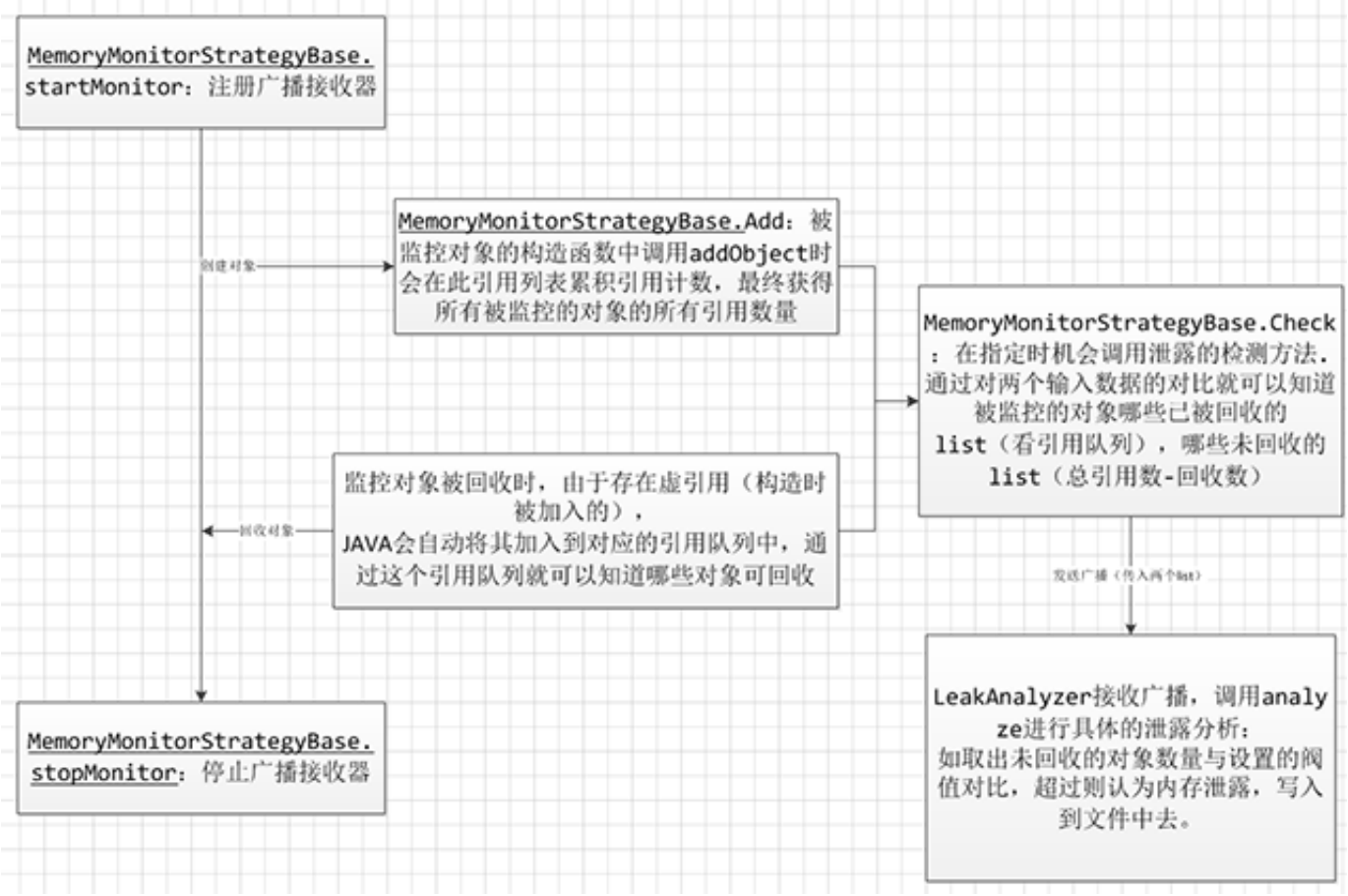
这是手机管家内的一个工具工程，正式打包不会打入，BVT等每日监控测试包可以打入。打入后可以通过诸如addObject接口（通过反射去检查是否含有该工具并调用）来加入需要监控的检测对象，这个工具会自动在指定时机（如退出管家）去检测该对象是否发生泄漏。

这个内存泄露检测的基本原理是：

虚引用主要用来跟踪对象被垃圾回收器回收的活动。虚引用必须和引用队列（ReferenceQueue）联合使用（在虚引用函数就必须关联指定）。当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，自动把这个虚引用加入到与之关联的引用队列中。程序可以通过判断引用队列中是否已经加入了虚引用，来了解被引用的对象是否将要被垃圾回收。

基于以上原理，MLD工具在调用接口addObject加入监控类型时，会为该类型对象增加一个虚引用，注意虚引用并不会影响该对象被正常回收。因此可以在ReferenceQueue引用队列中统计未被回收的监控对象是否超过指定阀值。

利用PhantomReferences(虚引用)和ReferenceQueue(引用队列)，当PhantomReferences被加入到相关联的ReferenceQueue时，则视该对象已经或处于垃圾回收器回收阶段了。



MLD监控原理核心

目前手机管家已对大部分类完成内存泄露的监控，包括各种activity，service和view页面等，务求在技术上能带给用户最顺滑的产品体验。

接下来简单介绍下这个工具的判断核心。根据虚引用监控到的内存状态，需要通过多种策略来判断是否存在内存泄露。

（1）最简单的方式就是直接在加入监控时就为该类型设定最大存在个数，举个例子，各个DAO对象理论上只能存在最多一个，因此一旦出现两个相同的DAO，那一般都是泄露了；

（2）第二种情况是在页面退出程序退出时，检索gc后无法释放的对象列表，这些对象类型也会成为内存泄露的怀疑对象；

（3）最后一种情况比较复杂，基本原理是根据历史操作判断对象数量的增长幅度。根据对象的增长通过最小二乘法拟合出该对象类型的增长速度，如果超过经验值则会列入疑似泄露的对象列表。

3.3 UIAutomator完成重复操作的自动化

最后一步就很简单了。这么多反复的UI操作，让人工来点就太浪费人力了。我们使用UIAutomator来进行自动化操作测试。

目前手机管家的每日自动化测试已覆盖各个功能的主路径，并通过配置文件的方式来灵活驱动用例的增删改查，最大限度保证了随着版本推移用例的复用价值。

至此手机管家的内存泄露测试方案介绍完毕，也欢迎各路牛人交流沟通更多更强的内存泄露工具盒方案！

编者按

下期精彩预告

解决问题还是要从源头抓起，了解了内存泄露的排查方法，肯定还有很多开发者想弄清内存泄露发生的本因。别着急，下周同一时间，Bugly将为大家推送内存泄露从入门到精通三部曲的下篇：内存泄露常见原因。敬请期待~