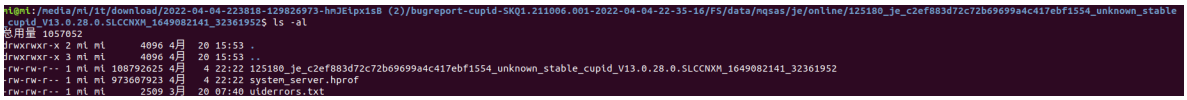
https://jira.n.xiaomi.com/browse/MIUIROM-634673

> 首先OOM 和内存泄漏是有区别的，不懂的可以自行百度；（面试题）

准备

- 既然是OOM，首先需要找到hropf文件，小米的hropf是放在bugreport的如下位置



- 然后就是要用到hprof分析工具，传统的是MAT，不过现在的Android Studio中预置的profiler也可以进行分析，不过打开比较卡，建议还是用MAT；
- MAT下载：https://www.eclipse.org/mat/ ，其中遇到的问题，是JDK版本低于11导致的，升到11以上就解决了；

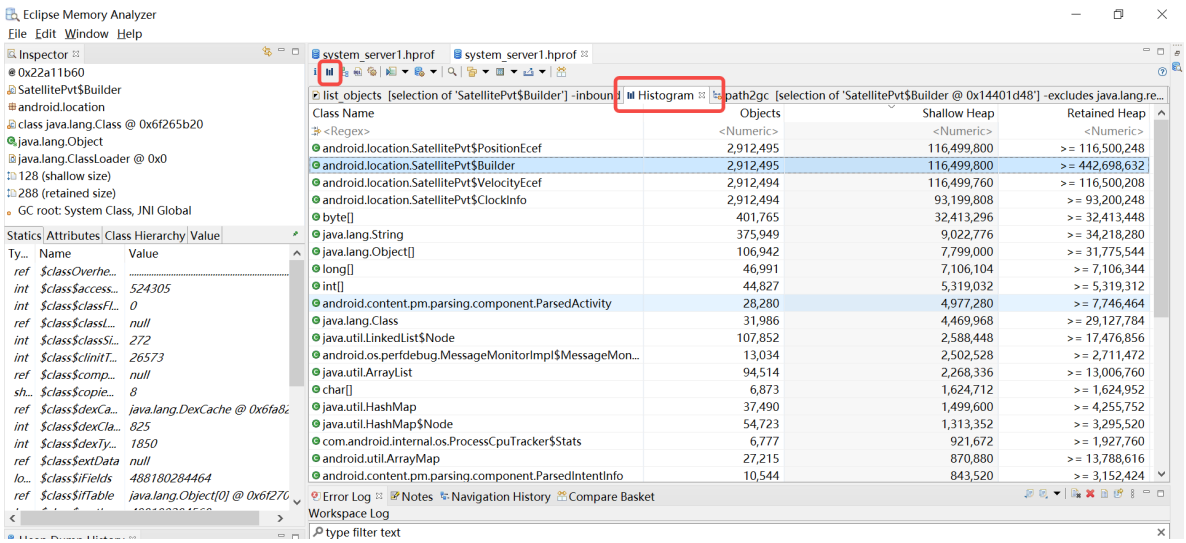
开始分析

1. **注意：Android 生成的 .hprof 文件在MAT上分析的时候需要进行转换一下格式：**

> 打开命令行窗口，在**android** SDK目录，执行以下命令：

> hprof-conv 1.hprof 2.hprof

1. **点击：Histogram**



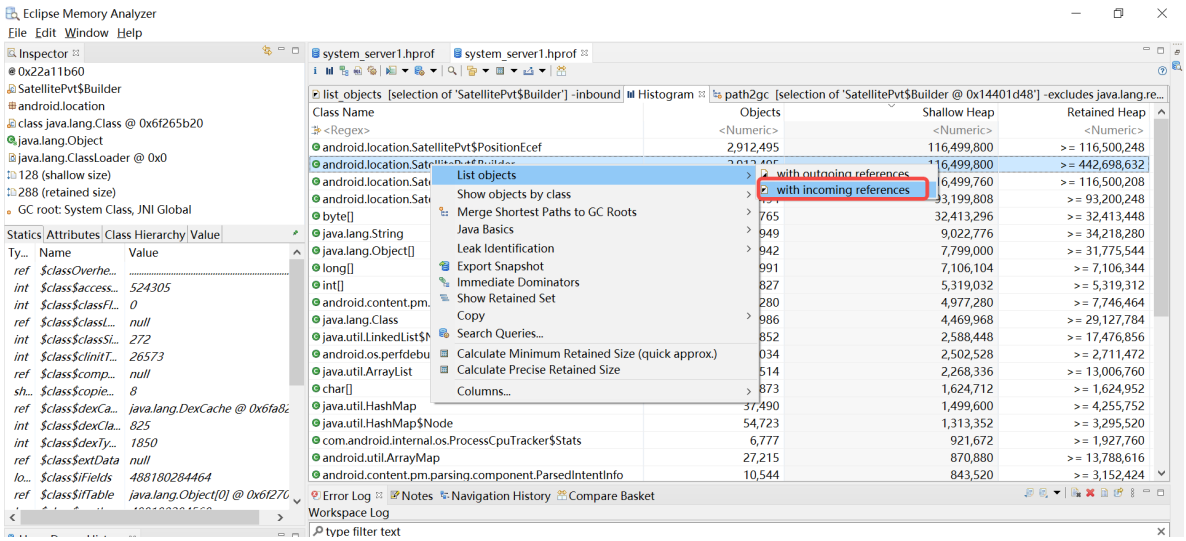
Histogram 可以列出内存中的对象，对象的个数及其内存大小，可以用来定位哪些对象在 Full GC 之后还活着，哪些对象占大部分内存。

- Class Name：类名称，Java 类名。
- Objects：类的对象的数量，这个对象被创建了多少个。
- Shallow Heap：对象本身占用内存的大小，不包含其引用的对象内存，实际分析中作用不大。常规对象(非数组)的 Shallow Size 由其成员变量的数量和类型决定。数组的 Shallow Size 由数组元

素的类型(对象类型、基本类型)和数组长度决定。对象成员都是些引用，真正的内存都在堆上，看起来是一堆原生的 byte[], char[], int[]，对象本身的内存都很小。

- Retained Heap：计算方式是将 Retained Set(当该对象被回收时那些将被 GC 回收的对象集合)中的所有对象大小叠加。或者说，因为 X 被释放，导致其它所有被释放对象(包括被递归释放的)所占的 heap 大小。Retained Heap 可以更精确的反映一个对象实际占用的大小。

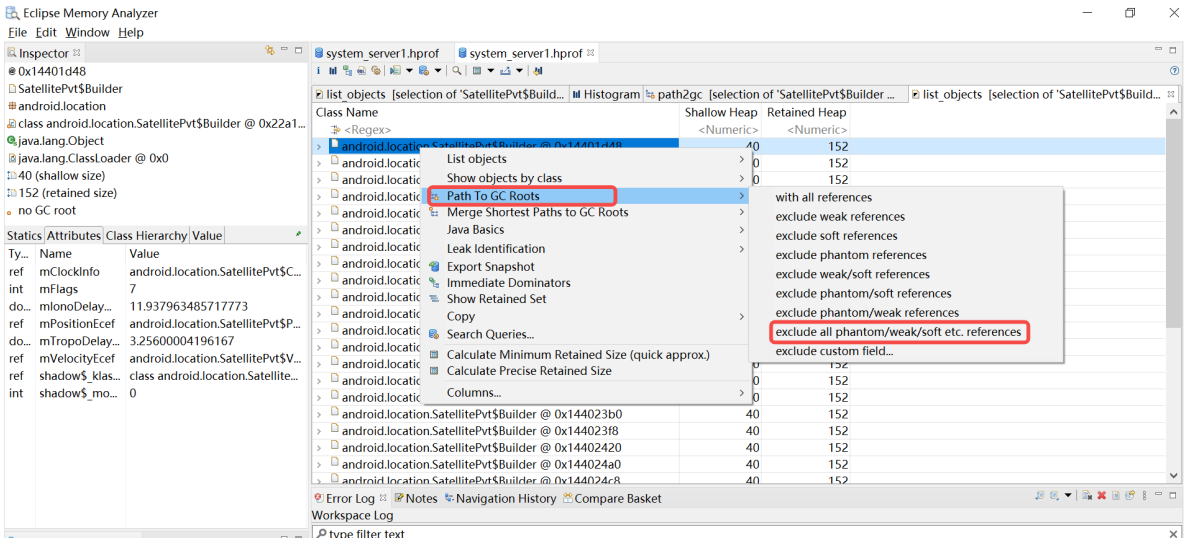
1. 查看调用栈



在上述列表中选择一个 Class，右键选择 List objects > with incoming references，在新页面会显示通过这个 class 创建的对象信息。

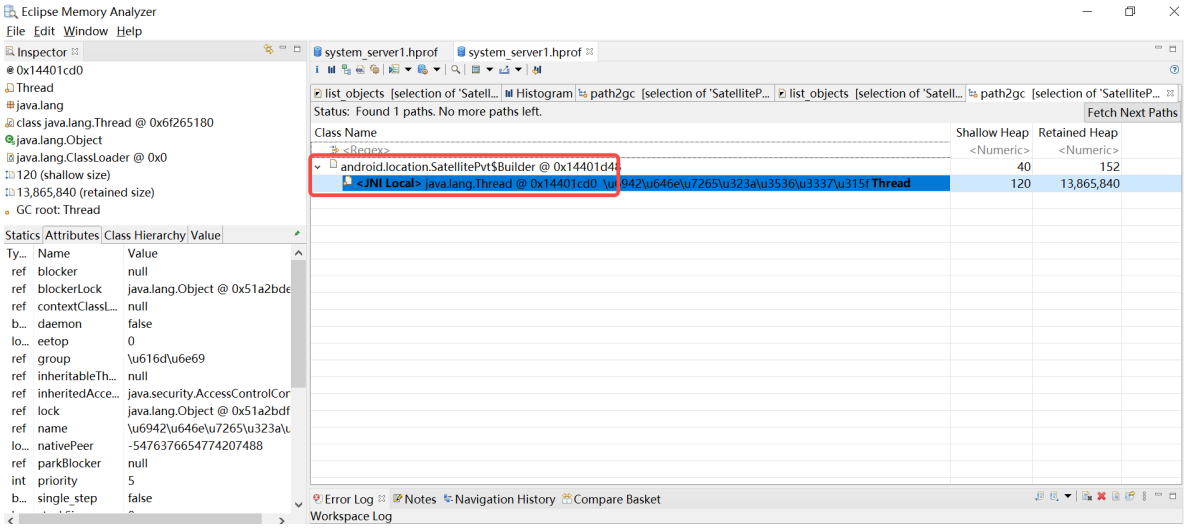
> with incoming references(即哪边创建了这个对象);
>
> with outcoming references(这个对象的实体类里面包含了哪些对象）；

1. 查看Strong reference的调用链



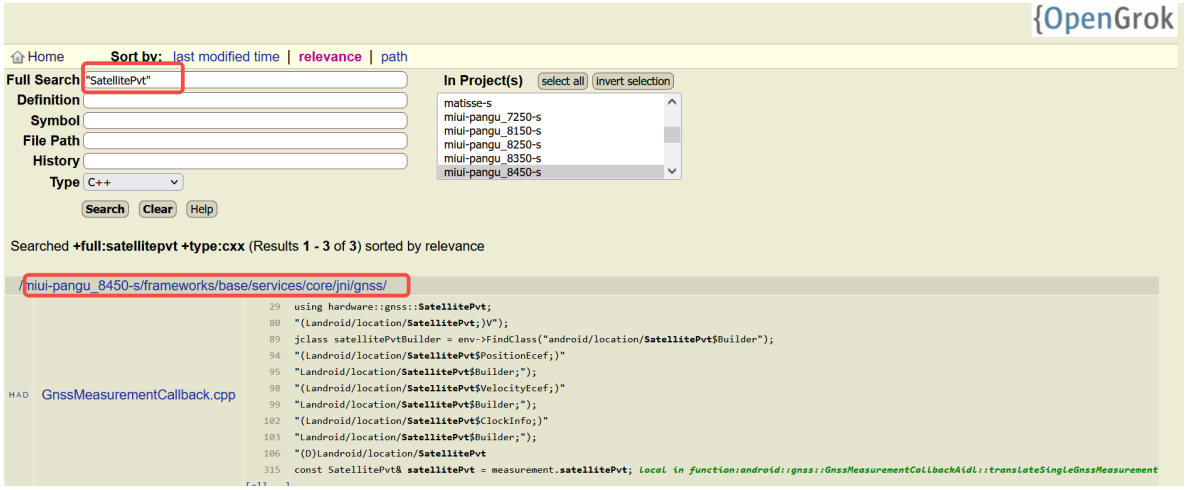
继续选择一个对象，右键选择 Path to GC Roots > ，通常在排查内存泄漏(一般是因为存在无效的引用)的时候，我们会选择 exclude all phantom/weak/soft etc.references，意思是查看排除虚引用/弱引用/软引用等的引用链，因为被虚引用/弱引用/软引用的对象可以直接被 GC 给回收，我们要看的就是某个对象否还存在 Strong 引用链(在导出 Heap Dump 之前要手动触发 GC 来保证)，如果有，则说明存在内存泄漏，然后再去排查具体引用。

这时会拿到 GC Roots 到该对象的路径，通过对象之间的引用，可以清楚的看出这个对象没有被回收的原因，然后再去定位问题。如果上面对象此时本来应该是被 GC 掉的，简单的办法就是将其中的某处置为 null 或者 remove 掉，使其到 GC Root 无路径可达，处于不可触及状态，垃圾回收器就可以回收了。反之，一个存在 GC Root 的对象是不会被垃圾回收器回收掉的。



1. 定位原因

a. 可以看到上面报到没有被回收的对象是在JNI里面不断的创建 `SatellitePvt builder` ,找到对应的JNI



b. 可以看到在这边不断的创建 `SatellitePvt builder`

```
    if (measurement.flags & static_cast<uint32_t>(GnssMeasurement::HAS_SATELLITE_PVT)) {
        const SatellitePvt& satellitePvt = measurement.satellitePvt;
        uint16_t satFlags = static_cast<uint16_t>(satellitePvt.flags);
        jobject positionEcef = nullptr;
        jobject velocityEcef = nullptr;
        jobject clockInfo = nullptr;
        jobject satellitePvtBuilderObject =
                env->NewObject(class_satellitePvtBuilder, method_satellitePvtBuilderCtor);
```

c. 这儿有个for循环,一次gnssMeasurementCb，会有多个data.measurements，要是一直在gnssMeasurementCb，那么就会一直会在处理，是指数级的增长; 然后gnssMeasurementCb是底层上报的，因此下面此问题甩给高通看；

```
414
415 jobjectArray GnssMeasurementCallbackAidl::translateAllGnssMeasurements(
416         JNIEnv* env, const std::vector<GnssMeasurement>& measurements) {
417     if (measurements.size() == 0) {
418         return nullptr;
419     }
420
421     jobjectArray gnssMeasurementArray =
422             env->NewObjectArray(measurements.size(), class_gnssMeasurement,
423                                 nullptr /* initialElement */);
424
425     for (uint16_t i = 0; i < measurements.size(); ++i) {
426         JavaObject object(env, class_gnssMeasurement, method_gnssMeasurementCtor);
427         translateSingleGnssMeasurement(env, measurements[i], object);
428         jobject gnssMeasurement = object.get();
429         env->SetObjectArrayElement(gnssMeasurementArray, i, gnssMeasurement);
430         env->DeleteLocalRef(gnssMeasurement);
431     }
432
```