HenCoder Plus 讲义

LeakCanary 源码解析

什么是内存泄露?

- 传统定义的内存泄露：申请的内存忘记释放了。
- Android （或 JVM）的内存泄露：短生命周期的对象被长生命周期的对象持有，从而导致短生命周 期的对象不能被释放

垃圾回收机制

垃圾回收机制分为「引用计数法」和「可达性分析法」：

- 「引用计数法」 `Python` , `Object-C` , `Swift`

  用一个计数器记录一个对象被引用的次数，如果引用的次数被减少到 0 那么说明这个对象是垃圾 对象。

  都是引用计数（引用计数有循环引用的问题）

- 「可达性分析法」 `Java`

  Jvm 通过一些 GC Roots 向下搜索，如果可以被 Gc Roots 引用到的对象，说明这个对象不是垃圾 对象，反之这个对象就算互相引用了也是垃圾对象 那些对象会被作为 GC Roots 呢?

  - 在线程栈中的局部变量，也就是正在被调用的方法，它里面的参数和局部变量
  - 存活的线程对象
  - JNI 的引用
  - Class 对象，因为 Android 加载 Class 后不会卸载 Class
  - 引用类型的静态变量

内存泄露的问题

内存泄漏并不会马上把程序搞挂掉。但是随着应用的使用，不能回收的垃圾对象会越来越多，就导致了 可用的内存越来越少，到最后应用就有可能在任何位置抛出 `OutOfMemoryError` ，这种情况下，每次 OOM 错误堆栈都不同，就很难定位问题。

监控 Activity 和 Fragment 原理

通过注册 `Application` 和 `Fragment` 上的生命周期回调来完成在 `Activity` 和 `Fragment` 销毁的时候开始观察。

```
public void registerActivityLifecycleCallbacks(
        @NonNull Application.ActivityLifecycleCallbacks callback) {
    synchronized (mActivityLifecycleCallbacks) {
        mActivityLifecycleCallbacks.add(callback);
    }
}
```

```
@Override
public void registerFragmentLifecycleCallbacks(@NonNull FragmentLifecycleCallbacks cb,
                                               boolean recursive) {
    mLifecycleCallbacks.add(new FragmentLifecycleCallbacksHolder(cb, recursive));
}
```

> 1.x 的 LeakCanary 只监控了 support 包和 API 26 以上的 Fragment 。

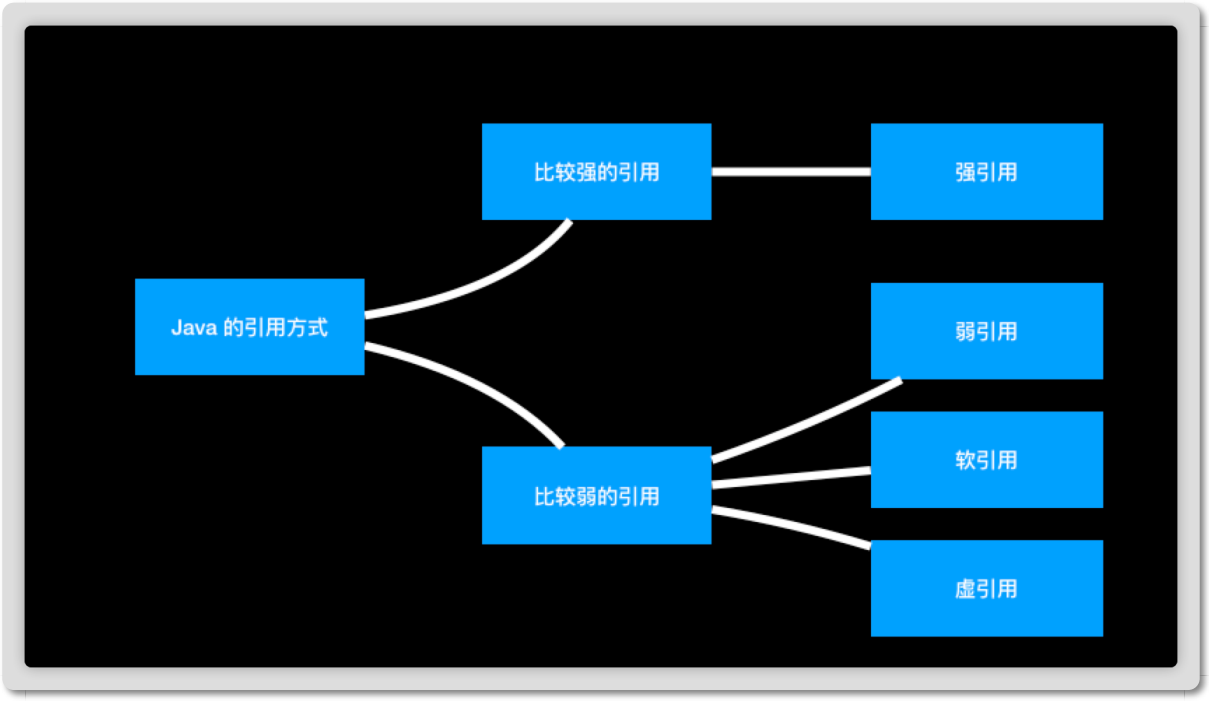
四大引用

- 强一点的引用

  强引用——不会被垃圾回收

- 弱一点的引用
  - 弱引用——可以通过 `get()` 获得引用对象，会被垃圾回收
  - 软引用——可以通过 `get()` 获得引用对象，内存不足会被垃圾回收
  - 虚引用——不能通过 `get()` 获得引用对象，会被垃圾回收

弱引用在引用对象被垃圾回收之前，会将引用放入它关联的队列中。所以可以通过队列中是否有对 应的引用来判断对象是否被垃圾回收了。



watch() 方法

原理：就是通过弱引用的方式来判断队列中是否有弱引用来判断对象是否被垃圾回收了。

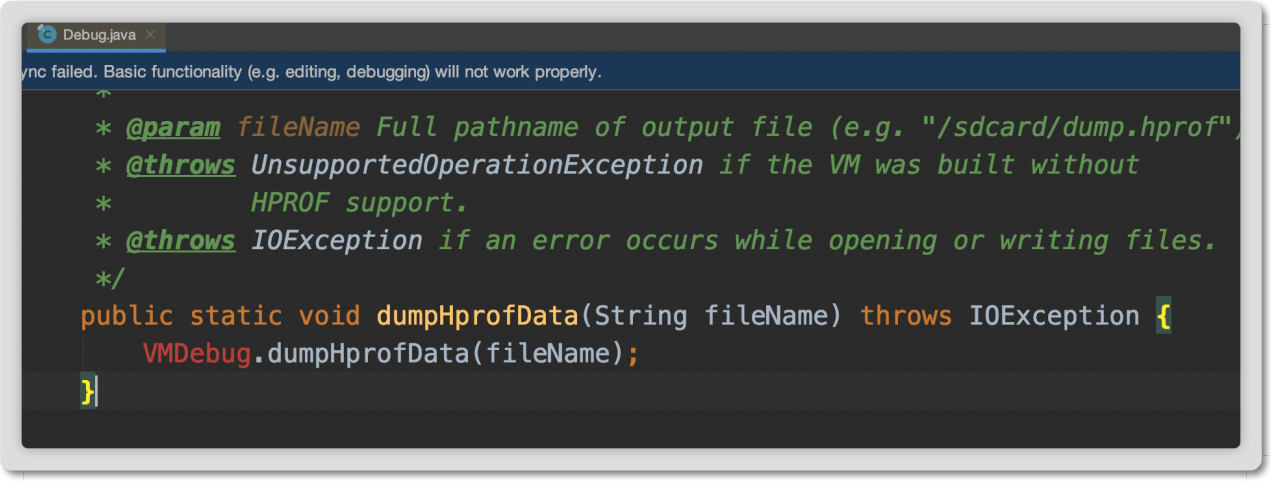
触发 GC 的正确姿势

```
public interface GcTrigger {
  GcTrigger DEFAULT = new GcTrigger() {
    @Override public void runGc() {
      // Code taken from AOSP FinalizationTest:
      // https://android.googlesource.com/platform/libcore/+/master/support/src/test/java/libcore/
      // java/lang/ref/FinalizationTester.java
      // System.gc() does not garbage collect every time. Runtime.gc() is
      // more likely to perform a gc.
      Runtime.getRuntime().gc();
      enqueueReferences();
      System.runFinalization();
    }

    private void enqueueReferences() {
      // Hack. We don't have a programmatic way to wait for the reference queue daemon to move
      // references to the appropriate queues.
      try {
        Thread.sleep( millis: 100);
      } catch (InterruptedException e) {
        throw new AssertionError();
      }
    }
  };

  void runGc();
}
```

通过 `dumpHprofData` 来获取 hprof 文件



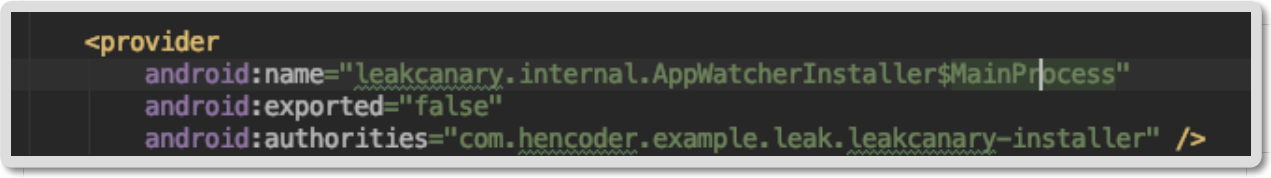
分析会在独立进程中进行，`1.x` 内部使用 `haha` ，`2.x` 内部使用 `shark` 。

在 `1.x` 版本中，通过 `haha` 第三方库进行堆文件分析。

在 `2.x` 版本中，通过 `shark` 第三方库进行堆文件分析，内存占用大幅度减少，分析速度大幅度提高。

2.0 不需要主动初始化的原理

`ContentProvider#onCreate` 会在 `Application#onCreate` 之前先执行，在这个 `onCreate` 中就可以进行初始化了。 LeakCanary 2.0 利用了这个 原理，所以不需要我们手动进行初始化



问题和建议?

课上技术相关的问题，都可以去群里和大家讨论，对于比较通用的、有价值的问题，可以去我们的知识星球提问。

具体技术之外的问题和建议，都可以找丢物线（微信：diuwuxian），丢丢会为你解答技术以外的一切。

觉得好?

如果你觉得课程很棒，欢迎给我们好评呀！https://ke.qq.com/comment/index.html?cid=381952

> 一定要是你真的觉得好，再给我们好评。不要仅仅因为对扔物线的支持而好评（报名课程已经是你最大的支持了，再不够的话 B 站多来点三连我也很开心），另外我们也坚决不做好评返现等任何的交易。我们只希望，在课程对你有帮助的前提下，可以看到你温暖的评价。

更多内容：

- 网站：https://hencoder.com；https://kaixue.io
- 各大搜索引擎、微信公众号、微博、知乎、掘金、哔哩哔哩、YouTube、西瓜视频、抖音、快手、微视：统一账号「**扔物线**」，我会持续输出优质的技术内容，欢迎大家关注。
- 哔哩哔哩快捷传送门：https://space.bilibili.com/27559447

大家如果喜欢我们的课程，还请去扔物线的哔哩哔哩，帮我素质三连，感谢大家！