**Java内存泄露——全解析和处理办法 [转载]**

@author 小筐子

@address *http://www.jianshu.com/p/bf159a9c391a*

JAVA是垃圾回收语言的一种，开发者无需特意管理内存分配。但是JAVA中还是存在着许多内存泄露的可能性，如果不好好处理内存泄露，会导致APP内存单元无法释放被浪费掉，最终导致内存全部占据**堆栈(heap)**挤爆进而**程序崩溃**。

**内存泄露**

说到**内存泄露**，就不得不提到**内存溢出**，这两个比较容易混淆的概念，我们来分析一下。

* **内存泄露**：**程序在向系统申请分配内存空间后(new)，在使用完毕后未释放。**结果导致一直占据该内存单元，我们和程序都无法再使用该内存单元，直到程序结束，这是内存泄露。
* **内存溢出**：**程序向系统申请的内存空间超出了系统能给的。**比如内存只能分配一个int类型，我却要塞给他一个long类型，系统就出现oom。又比如一车最多能坐5个人，你却非要塞下10个，车就挤爆了。

大量的内存泄露会导致内存溢出(oom)。

**内存**

想要了解内存泄露，对内存的了解必不可少。  
JAVA是在JVM所虚拟出的内存环境中运行的，JVM的内存可分为三个区：堆(heap)、栈(stack)和方法区(method)。

* **栈(stack)**：是简单的数据结构，但在计算机中使用广泛。栈最显著的特征是：**LIFO(Last In, First Out, 后进先出)**。比如我们往箱子里面放衣服，先放入的在最下方，只有拿出后来放入的才能拿到下方的衣服。栈中只存放基本类型和对象的引用(不是对象)。
* **堆(heap)**：**堆内存用于存放由new创建的对象和数组**。在堆中分配的内存，由java虚拟机自动垃圾回收器来管理。JVM只有一个堆区(heap)被所有线程共享，堆中不存放基本类型和对象引用，只存放对象本身。
* **方法区(method)**：又叫静态区，跟堆一样，被所有的线程共享。方法区包含所有的class和static变量。

内存的概念大概理解清楚后，要考虑的问题来了：  
**到底是哪里的内存会让我们造成内存泄露？**

**内存泄露原因分析**

在JAVA中JVM的栈记录了方法的调用，每个线程拥有一个栈。在线程的运行过程当中，执行到一个新的方法调用，就在栈中增加一个内存单元，即帧(frame)。在frame中，保存有该方法调用的参数、局部变量和返回地址。然而JAVA中的局部变量只能是基本类型变量(int)，或者对象的引用。**所以在栈中只存放基本类型变量和对象的引用。**引用的对象保存在堆中。

**当某方法运行结束时，该方法对应的frame将会从栈中删除，frame中所有局部变量和参数所占有的空间也随之释放。线程回到原方法继续执行，当所有的栈都清空的时候，程序也就随之运行结束。**

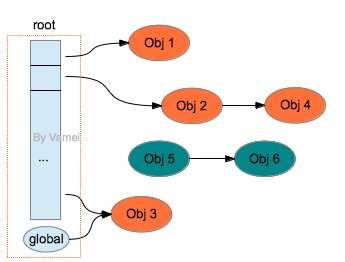
而对于堆内存，堆存放着普通变量。**在JAVA中堆内存不会随着方法的结束而清空**，所以在方法中定义了局部变量，在方法结束后变量依然存活在堆中。

综上所述，栈(stack)可以自行清除不用的内存空间。但是如果我们不停的创建新对象，堆(heap)的内存空间就会被消耗尽。所以JAVA引入了垃圾回收(garbage collection，简称GC)去处理堆内存的回收，但如果对象一直被引用无法被回收，造成内存的浪费，无法再被使用。所以**对象无法被GC回收**就是造成内存泄露的原因！

**垃圾回收机制**

垃圾回收(garbage collection，简称GC)可以自动清空堆中不再使用的对象。在JAVA中对象是通过引用使用的。如果再没有引用指向该对象，那么该对象就无从处理或调用该对象，这样的对象称为**不可到达**（unreachable）。**垃圾回收用于释放不可到达的对象所占据的内存。**

实现思想：我们将栈定义为root，遍历栈中所有的对象的引用，再遍历一遍堆中的对象。因为栈中的对象的引用执行完毕就删除，所以我们就可以通过栈中的对象的引用，查找到堆中没有被指向的对象，这些对象即为不可到达对象，对其进行垃圾回收。



垃圾回收实现思想

**如果持有对象的强引用，垃圾回收器是无法在内存中回收这个对象。**

**引用类型**

在JDK 1.2以前的版本中，若一个对象不被任何变量引用，那么程序就无法再使用这个对象。也就是说，只有对象处于可触及(reachable)状态，程序才能使用它。从JDK 1.2版本开始，把对象的引用分为4种级别，从而使程序能更加灵活地控制对象的生命周期。这4种级别由高到低依次为：强引用、软引用、弱引用和虚引用。  
[Java/Android引用类型及其使用分析](http://www.cnblogs.com/lwbqqyumidi/p/4151833.html)

**1. 强引用(Strong reference)**  
实际编码中最常见的一种引用类型。常见形式如：A a = new A();等。强引用本身存储在栈内存中，其存储指向对内存中对象的地址。一般情况下，当对内存中的对象不再有任何强引用指向它时，垃圾回收机器开始考虑可能要对此内存进行的垃圾回收。如当进行编码：a = null，此时，刚刚在堆中分配地址并新建的a对象没有其他的任何引用，当系统进行垃圾回收时，堆内存将被垃圾回收。

**2. 软引用(Soft Reference)**  
软引用的一般使用形式如下：

A a = new A();  
SoftReference<A> srA = new SoftReference<A>(a);

软引用所指示的对象进行垃圾回收需要满足如下两个条件：  
1.当其指示的对象没有任何强引用对象指向它；  
2.当虚拟机内存不足时。  
因此，SoftReference变相的延长了其指示对象占据堆内存的时间，直到虚拟机内存不足时垃圾回收器才回收此堆内存空间。

**3. 弱引用(Weak Reference)**  
同样的，软引用的一般使用形式如下：

A a = new A();  
WeakReference<A> wrA = new WeakReference<A>(a);

WeakReference不改变原有强引用对象的垃圾回收时机，一旦其指示对象没有任何强引用对象时，此对象即进入正常的垃圾回收流程。

**4. 虚引用(Phantom Reference)**  
与SoftReference或WeakReference相比，PhantomReference主要差别体现在如下几点：  
1.PhantomReference只有一个构造函数

PhantomReference(T referent, ReferenceQueue<? super T> q)

2.不管有无强引用指向PhantomReference的指示对象，PhantomReference的get()方法返回结果都是null。

因此，PhantomReference使用必须结合ReferenceQueue；  
与WeakReference相同，PhantomReference并不会改变其指示对象的垃圾回收时机。

**内存泄露原因**

如果持有对象的强引用，垃圾回收器是无法在内存中回收这个对象。

内存泄露的真因是：**持有对象的强引用，且没有及时释放，进而造成内存单元一直被占用，浪费空间，甚至可能造成内存溢出！**

**其实在Android中会造成内存泄露的情景无外乎两种：**

* 全局进程(process-global)的static变量。这个无视应用的状态，持有Activity的强引用的怪物。
* 活在Activity生命周期之外的线程。没有清空对Activity的强引用。

检查一下你的项目中是否有以下几种情况：

* Static Activities
* Static Views
* Inner Classes
* Anonymous Classes
* Handler
* Threads
* TimerTask
* Sensor Manager

详解见该文章[《Android内存泄漏的八种可能》](http://www.jianshu.com/p/ac00e370f83d?hmsr=toutiao.io&utm_medium=toutiao.io&utm_source=toutiao.io)

最后推荐一个可检测app内存泄露的项目：[LeakCanary](https://github.com/square/leakcanary)（可以检测app的内存泄露）