**让人蛋疼的JAVA虚引用！**

<https://mp.weixin.qq.com/s/V62uA82vhS0hZFkyX5VHYw>

在Java的世界里，对象的存在层次，也有三六九等，充满了阶层之间的嘲弄。强软弱虚各种引用，对于熟悉Java的同学一定不会感到陌生，它们随着等级的降低，越来越没存在感。平常使用的对象，大多数就是强引用的；而软引用和弱引用，则经常在一些堆内缓存框架中用到。

那虚引用呢？传说中的幽灵引用，是不是就如同它的名字一样，一无是处呢？

**三种引用**

首先，我们来回顾一下其他三种引用的类型和用途。

**Strong references**

当内存空间不足，系统撑不住了，JVM 就会抛出 OutOfMemoryError 错误。即使程序会异常终止，这种对象也不会被回收。这种引用属于最普通最强硬的一种存在，只有在和 GC Roots 断绝关系时，才会被消灭掉。

这种引用，你每天的编码都在用。例如：new 一个普通的对象。



这种方式可能是有问题的。假如你的系统被大量用户（User）访问，你需要记录这个 User 访问的时间。可惜的是，User 对象里并没有这个字段，所以我们决定将这些信息额外开辟一个空间进行存放。

### Soft references

软引用用于维护一些可有可无的对象。在内存足够的时候，软引用对象不会被回收，只有在内存不足时，系统则会回收软引用对象，如果回收了软引用对象之后仍然没有足够的内存，才会抛出内存溢出异常。

可以看到，这种特性非常适合用在缓存技术上。比如网页缓存、图片缓存等。

Guava 的 CacheBuilder，就提供了软引用和弱引用的设置方式。在这种场景中，软引用比强引用安全的多。

软引用可以和一个引用队列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收，Java 虚拟机就会把这个软引用加入到与之关联的引用队列中。

### Weak references

弱引用对象相比较软引用，要更加无用一些，它拥有更短的生命周期。

当 JVM 进行垃圾回收时，无论内存是否充足，都会回收被弱引用关联的对象。弱引用拥有更短的生命周期，在 Java 中，用 java.lang.ref.WeakReference 类来表示。

## 怪异的虚引用

以上几个引用级别都很好理解，但是虚引用是个例外。虚引用可以使用下面的代码定义：



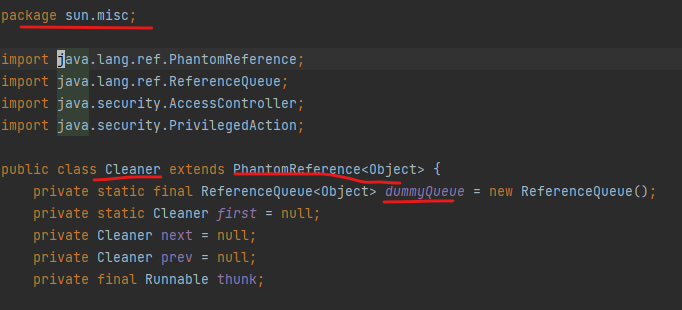
**虚引用主要用来跟踪对象被垃圾回收的活动。**

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，就会在回收对象之前，把这个虚引用加入到与之关联的引用队列中。

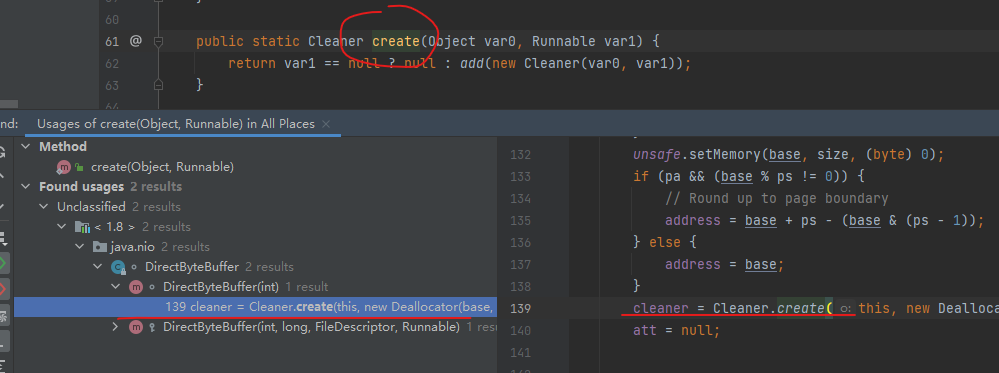
程序如果发现某个虚引用已经被加入到引用队列，那么就可以在所引用的对象的内存被回收之前采取必要的行动。

## 桃花源深处

在hotspot的jvm中，有一个叫做cleaner的类，其实就是虚引用典型的应用。可以看到Cleaner是直接简单粗暴的继承了PhantomReference，所以它本质上就是一个虚引用，只不过多了一些便捷的操作。

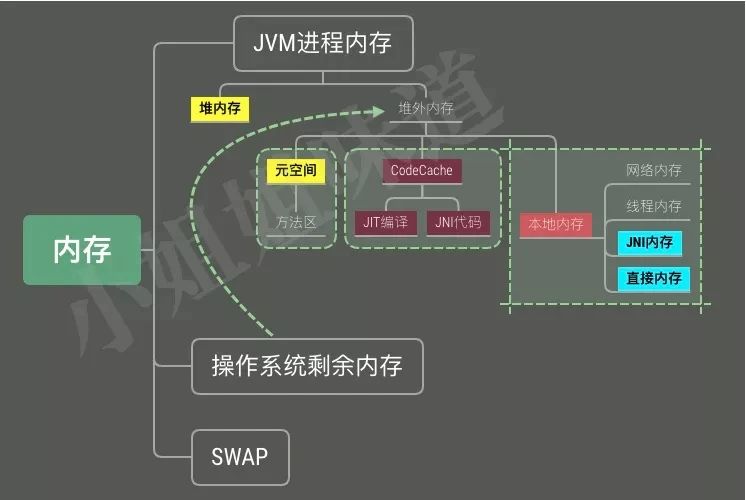


那么这个类是在什么地方用到的呢？大家手上应该都有jdk的源代码，追踪一下，发现最后竟然是DirectByteBuffer用到了它。



直接内存，一直是一个看起来非常高大上的名词，基本上和高性能挂钩，但也容易产生内存泄漏。由于直接内存，是属于堆外内存的，所以垃圾回收的时候，就不能靠JVM的那一套垃圾回收算法进行清理。

事实上，由于DirectByteBuffer可能会被使用较长时间，熬过了年轻代的各种回收，就会进入老年代。这时候就比较麻烦了，这些引用对象，要在下一轮Old GC或者Full GC才能触发，如果你的老年代空间较大，触发回收的操作就需要等很久很久。问题是，在这段时间内，虽然这些堆外内存不再使用了，但它仍然占用着较大的物理空间，最后造成严重的浪费甚至崩溃。



那么这些堆外内存是如何进行回收的呢？这就是Cleaner的作用。Cleaner通过next和prev构造了一个典型的链表，但它本身是没有任何逻辑的，因为它的清理逻辑都在thunk方法中。