参考：

Java核心36讲

前言

引用是Java备受关注的一点，我们日常用到最多的就是强引用，其实Java一共提供了四个引用：强引用，软引用，弱引用，虚引用。搞这么多引用到底有啥含义呢，其实这就要结合JVM来看了。掌握了引用对于掌握Java对象生命周期以及JVM的相关机制是有帮助的。但毕竟引用本身还是涉及在我们的Java代码层，所以我放在这里叙述，没有归类到JVM。

白话理解

为何说我们日常用的最多的就是强引用呢，举个例子：

```
Object object = new Object();
```

这样的就是一个标准的强引用，只要强引用还在，GC就不会碰这个对象，所以强引用能活到JVM终止。当然只要超出了引用的作用域（方法中的局部强引用，方法结束也就结束了）或显示的这样

```
object = null;
```

GC就能回收了。也正因为强引用的倔强，大数据量的单次操作，操作不当很容易抛出OOM。

软引用则显得灵活多了，软引用的对象对于JVM来说是可有可无的，用古话说就是“食之无味，弃之可惜”，一般来说这样的东西我们都会选择先放着，然后实在放不下了就扔掉，软引用就是这样，一旦JVM内存不够就会执行GC把软引用对象清除。

弱引用就更显得可有可无了，任何时候的GC都会把它直接清除。

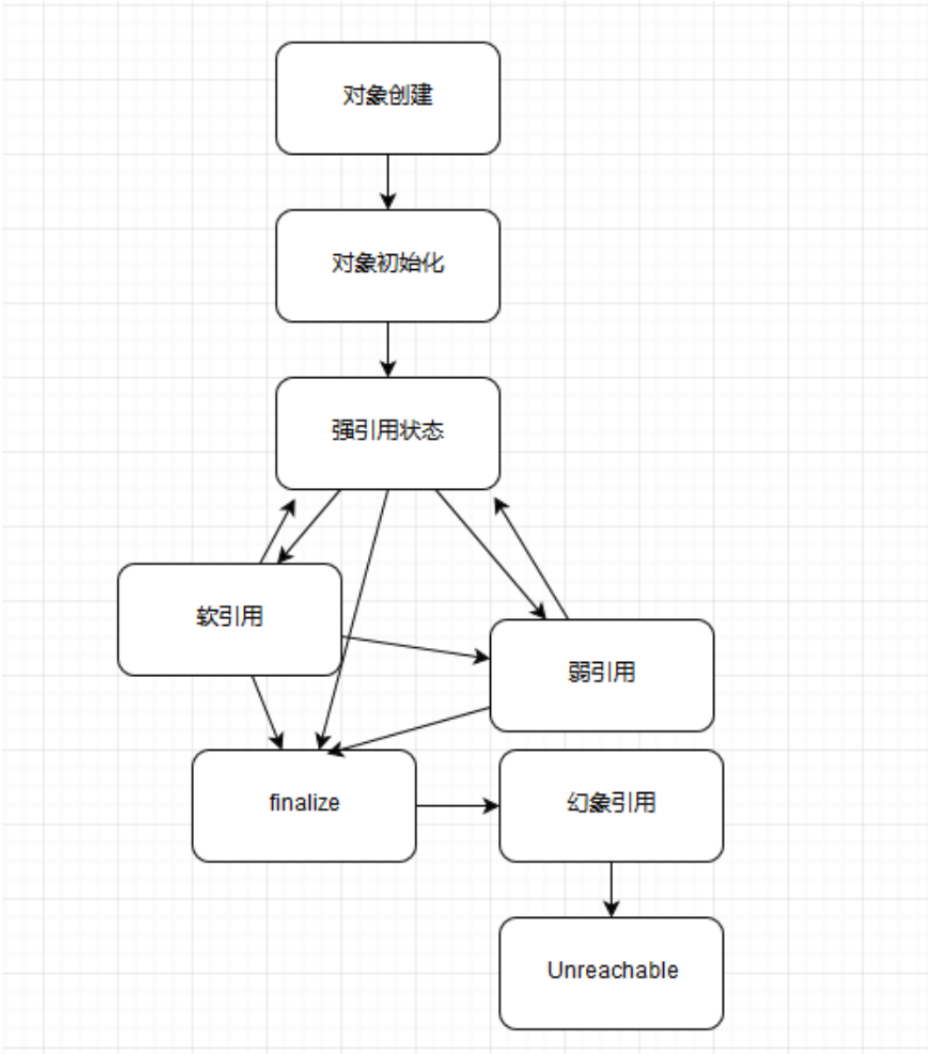
虚引用又称幻想引用，你不能通过它访问对象，换句话说，你永远访问不了虚引用对象，它主要提供一种确保对象呗finalize后做某些事情的机制，比如Post-Mortem清理机制，还有监控对象的创建和销毁。

总结如下。

引用类型	被垃圾回收时间	用途	生存时间
强引用	从来不会	对象的一般状态	JVM停止运行时终止
软引用	在内存不足时	对象缓存	内存不足时终止
弱引用	在垃圾回收时	对象缓存	垃圾回收时终止
虚引用	Unkonwn	Unkonwn	Unkonwn

可达性解读

不同的引用方式在Java中是有可以转化和不能转化的情况的，例如强引用转软引用，软引用转强引用这样的，这称作可达性。下图展示了可达性。



根据该图，我们总结下Java定义的可达性级别。

强可达，一个或多个线程可以不通过各种引用访问到一个对象的情况，比如，新创建一个对象，创建该对象的线程对该对象就是强可达；

软可达，只能通过软引用才能访问到对象的状态；

弱可达，只能通过弱引用才能访问到对象的状态，当弱引用清除时，就符合finalize条件了；

幻想可达，只有幻想引用指向该对象的时候；

不可达，对象可以被清除了。

所有引用类型均是抽象类 java.lang.ref.Reference的子类，它提供了get()方法，除了虚引用（只会获取到null），软引用和弱引用，只要对象还未销毁，都可以j将访问的对象重新指向强引用，这就是图

中双向箭头的含义。所以，针对软引用和弱引用，GC可能会存在二次确认，确保并未将对象改为强引用。

这也是某些情况出问题的源头了，如果我们错误的保持了强引用（赋值给了static变量），那么对象可能就没有变回类似弱引用的可达性状态了，就会产生内存泄漏。所以检查弱引用指向对象是否被h回收，也是诊断内存泄漏的一个思路。

诊断JVM引用情况

当要检查应用是否存在引用导致的回收问题，你可以选择HotSpot JVM自带的选项（PrintReferenceGC）去获取信息，比如在JDK8中，你可以在启动参数加入以下语句：

-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintReferenceGC

这样执行GC的时候就能打印引用的详细信息了，比如，这边我们手动System.gc();就可以在控制台看到这个：

```
0.197: [GC (System.gc()) 0.198: [SoftReference, 0 refs, 0.0000325 secs]0.198: [WeakReference, 10 refs, 0.0000219 secs]0.198: [FinalReference, 16 refs, 0.0000385 secs]0.198: [PhantomReference, 0 refs, 0 refs, 0.0000238 secs]0.198: [JNI Weak Reference, 0.0000136 secs][PSYoungGen: 2621K->728K(76288K)] 2621K->736K(251392K), 0.0016213 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
0.198: [Full GC (System.gc()) 0.199: [SoftReference, 0 refs, 0.0000257 secs]0.199: [WeakReference, 3 refs, 0.0000117 secs]0.199: [FinalReference, 0 refs, 0.0000200 secs]0.199: [PhantomReference, 0 refs, 0 refs, 0.0000302 secs]0.199: [JNI Weak Reference, 0.0000079 secs][PSYoungGen: 728K->0K(76288K)] [ParOldGen: 8K->595K(175104K)] 736K->595K(251392K), [Metaspace: 2790K->2790K(1056768K)], 0.0048867 secs] [Times: user=0.00 sys=0.00, real=0.01 secs
```

各自引用的信息可以说是一目了然的。