JVM之对象引用的四种类型：强软弱虚

在java中，一个对象的引用其实是分为好几种的，每种类型意味着这个对象具体会在什么时候会被回收。

对象的引用根据它被垃圾回收的时机，可以分为强、软、弱、虚四种。

我们日常中不了解这个的时候，自己定义的引用类型变量基本都是属于强引用。

接下来分别使用过四段代码来分别演示这四种类型：

|  |
| --- |
| import org.junit.Test;  import java.io.IOException;  import java.lang.ref.\*;  import java.util.ArrayList;  import java.util.List;  public class RefTest {  /\*\*  \* 强引用， 只有没有引用指向它时，才会被垃圾回收  \*  \* @throws IOException  \*/  @Test  public void test1() throws IOException {  RefTTT refTTT = new RefTTT();  refTTT = null;  System.gc();  System.in.read();  }  /\*\*  \* 软引用， 当需要分配新内存时，内存不够，触发垃圾回收时，会将软引用占用的内存进行回收。（应用场景：缓存）  \*/  @Test  public void test2() {  SoftReference<byte[]> softReference = new SoftReference<>(new byte[1024 \* 1024 \* 10]);  System.out.println(softReference.get());  try {  Thread.sleep(500);  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println(softReference.get());  byte[] bytes = new byte[1024 \* 1024 \* 8];  System.out.println(softReference.get());  }  /\*\*  \* 弱引用， 只要垃圾回收触发，就会将弱引用内存进行回收（应用场景：ThreadLocal）  \*/  @Test  public void test3() {  WeakReference<RefTTT> weakReference = new WeakReference<>(new RefTTT());  System.out.println(weakReference.get());  System.gc();  System.out.println(weakReference.get());  }  /\*\*  \* 虚引用， 虚引用只要垃圾回收，自然随之回收。  \* 虚引用的对象，是get不到的  \* 当对象被垃圾回收的时候，会将该对象加入到ReferenceQueue中  \* 用途：用于检测对象被垃圾回收时做出某些操作，例如当一个对象的内存不是分配在堆内，而是堆外内存，此时当检测到对该对象进行了垃圾回收时，可以进行堆外内存的清理操作。  \* 如果引用对象对finalize方法进行了重写，则需要进行两次gc才能加入到队列中。因为第一次GC时，finalize方法没有被执行过，所以即使对象已经没有其他引用了，  \* 但是仍然不满足phantom可达对象的条件（对象处于finalized状态）。  \* 如果对象没有覆盖finalize方法就可以直接视为finalized状态，那么一次GC就可以将对象视为phantom可达对象。  \* （phantom可达对象：对象不是强，软，弱可达，同时被设置为finalized后（finalized状态：对象的finalize方法执行后，对象所处的状态），对象就是phantom可达）  \* 第二次GC发现对象没有其他引用，同时又处于finalized状态，那么这次就可以将对象视为phantom可达对象。（会被加入到队列）  \*  \*/  /\*\*  \* Unfinalized: 对象被创建后，而且有比phantom reference更强的引用。  \* Finalizable： GC发现对象只有phantom reference引用后，且其finalize方法没有执行。  \* Finalized：无论对象有没有引用，只要对象的fanalize方法被执行，对象处于Finalized状态  \* Reclaimed: GC释放了对象占用的内存  \*/  @Test  public void test4() throws IOException, InterruptedException {  List<byte[]> list = new ArrayList<>();  ReferenceQueue<RefTTT> queue = new ReferenceQueue<>();  new Thread(() -> {  while (true) {  try {  Reference<? extends RefTTT> poll = queue.remove();  System.out.println(poll);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }).start();  PhantomReference<RefTTT> p = new PhantomReference<>(new RefTTT(), queue);  System.out.println(p.get());  //如果没有重写 finalize 只需要进行一次gc  System.gc();  Thread.sleep(2000);  System.gc();  // new Thread(()->{  // while (true){  // list.add(new byte[1024\*1024]);  // System.out.println("add 1 M");  // }  // }).start();  System.in.read();  }  } |