Java理论与实践:用弱引用堵住内存泄漏 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022_Java_E7_90_86_E8_AE_BA_c104_252437.htm 虽然用 Java&#8482. 语言编写的程序在理论上是不会出现“ 内存泄漏 ”的，但是有时对象在不再作为程序的逻辑状态的一部分之后仍然不被垃圾收集。本月，负责保障应用程序健康的工程师 Brian Goetz 探讨了无意识的对象保留的常见原因，并展示了如何用弱引用堵住泄漏。 要让垃圾收集（GC）回收程序不再使用的对象，对象的逻辑 生命周期（应用程序使用它的时间）和对该对象拥有的引用的实际 生命周期必须是相同的。在大多数时候，好的软件工程技术保证这是自动实现的，不用我们对对象生命周期问题花费过多心思。但是偶尔我们会创建一个引用，它在内存中包含对象的时间比我们预期的要长得多，这种情况称为无意识的对象保留（unintentional object retention）。 全局 Map 造成的内存泄漏 无意识对象保留最常见的原因是使用 Map 将元数据与临时对象（transient object）相关联。假定一个对象具有中等生命周期，比分配它的那个方法调用的生命周期长，但是比应用程序的生命周期短，如客户机的套接字连接。需要将一些元数据与这个套接字关联，如生成连接的用户的标识。在创建 Socket 时是不知道这些信息的，并且不能将数据添加到 Socket 对象上，因为不能控制 Socket 类或者它的子类。这时，典型的方法就是在一个全局 Map 中存储这些信息，如清单 1 中的 SocketManager 类所示： 清单 1. 使用一个全局 Map 将元数据关联到一个对象public class SocketManager { private Map m = new HashMap(). public void

setUser(Socket s, User u) { m.put(s, u). } public User getUser(Socket s) { return m.get(s). } public void removeUser(Socket s) { m.remove(s). }}SocketManager socketManager....socketManager.setUser(socket, user). 这种方法的问题是元数据的生命周期需要与套接字的生命周期挂钩，但是除非准确地知道什么时候程序不再需要这个套接字，并记住从 Map 中删除相应的映射，否则，Socket 和 User 对象将会永远留在 Map 中，远远超过响应了请求和关闭套接字的时间。这会阻止 Socket 和 User 对象被垃圾收集，即使应用程序不会再使用它们。这些对象留下来不受控制，很容易造成程序在长时间运行后内存爆满。除了最简单的情况，在几乎所有情况下找出什么时候 Socket 不再被程序使用是一件很烦人和容易出错的任务，需要人工对内存进行管理。 找出内存泄漏程序有内存泄漏的第一个迹象通常是它抛出一个 OutOfMemoryError，或者因为频繁的垃圾收集而表现出糟糕的性能。幸运的是，垃圾收集可以提供能够用来诊断内存泄漏的大量信息。如果以 -verbose:gc 或者 -Xloggc 选项调用 JVM，那么每次 GC 运行时在控制台上或者日志文件中会打印出一个诊断信息，包括它所花费的时间、当前堆使用情况以及恢复了多少内存。记录 GC 使用情况并不具有干扰性，因此如果需要分析内存问题或者调优垃圾收集器，在生产环境中默认启用 GC 日志是值得的。 有工具可以利用 GC 日志输出并以图形方式将它显示出来，JTune 就是这样的一种工具（请参阅 参考资料）。观察 GC 之后堆大小的图，可以看到程序内存使用的趋势。对于大多数程序来说，可以将内存使用分为两部分：baseline 使用和 current load 使用。对于服务器应用

程序，baseline 使用就是应用程序在没有任何负荷、但是已经准备好接受请求时的内存使用，current load 使用是在处理请求过程中使用的、但是在请求处理完成后会释放的内存。只要负荷大体上是恒定的，应用程序通常会很快达到一个稳定的内存使用水平。如果在应用程序已经完成了其初始化并且负荷没有增加的情况下，内存使用持续增加，那么程序就可能在处理前面的请求时保留了生成的对象。 清单 2 展示了一个有内存泄漏的程序。MapLeaker 在线程池中处理任务，并在一个 Map 中记录每一项任务的状态。不幸的是，在任务完成后它不会删除那一项，因此状态项和任务对象（以及它们的内部状态）会不断地积累。 清单 2. 具有基于 Map 的内存泄漏的程序public class MapLeaker { public ExecutorService exec = Executors.newFixedThreadPool(5). public Map taskStatus = Collections.synchronizedMap(new HashMap()). private Random random = new Random(). private enum TaskStatus { NOT_STARTED, STARTED, FINISHED }. private class Task implements Runnable { private int[] numbers = new int[random.nextInt(200)]. public void run() { int[] temp = new int[random.nextInt(10000)]. taskStatus.put(this, TaskStatus.STARTED). doSomeWork(). taskStatus.put(this, TaskStatus.FINISHED). } } public Task newTask() { Task t = new Task(). taskStatus.put(t, TaskStatus.NOT_STARTED). exec.execute(t). return t. }}