java中垃圾回收机制(GC)Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022_java_E4_B8_AD_E5_9E_83_c104_644701.htm 在java语中GC 即垃圾收集机制是指jvm用于释放那些不再使用的对象所占用的内存。java语言并不要求jvm有gc，也没有规定gc如何工作。不过常用的jvm都有gc，而且大多数gc都使用类似的算法管理内存和执行收集操作。 垃圾收集的目的在于清除不再使用的对象。gc通过确定对象是否被活动对象引用来确定是否收集该对象。gc首先要判断该对象是否是时候可以收集。两种常用的方法是引用计数和对象引用遍历。 1.1.引用计数 引用计数存储对特定对象的所有引用数，也就是说，当应用程序创建引用以及引用超出范围时，jvm必须适当增减引用数。当某对象的引用数为0时，便可以进行垃圾收集。 1.2.对象引用遍历 早期的jvm使用引用计数，现在大多数jvm采用对象引用遍历。对象引用遍历从一组对象开始，沿着整个对象图上的每条链接，递归确定可到达（reachable）的对象。如果某对象不能从这些根对象的一个（至少一个）到达，则将它作为垃圾收集。在对象遍历阶段，gc必须记住哪些对象可以到达，以便删除不可到达的对象，这称为标记（marking）对象。 然后，gc要删除不可到达的对象。删除时，有些gc只是简单的扫描堆栈，删除未标记的对象，并释放它们的内存以生成新的对象，这叫做清除（sweeping）。这种方法的问题在于内存会分成好多小段，而它们不足以用于新的对象，但是组合起来却很大。因此，许多gc可以重新组织内存中的对象，并进行压缩（compact），形成可利用的空间。 为此，gc需要停止其他的

活动活动。这种方法意味着所有与应用程序相关的工作停止，只有gc运行。结果，在响应期间增减了许多混杂请求。另外，更复杂的 gc不断增加或同时运行以减少或者清除应用程序的中断。有的gc使用单线程完成这项工作，有的则采用多线程以增加效率。 几种垃圾回收机制 2.1.标记－清除收集器 这种收集器首先遍历对象图并标记可到达的对象，然后扫描堆栈以寻找未标记对象并释放它们的内存。这种收集器一般使用单线程工作并停止其他操作。 2.2.标记－压缩收集器 有时也叫标记－清除－压缩收集器，与标记－清除收集器有相同的标记阶段。在第二阶段，则把标记对象复制到堆栈的新域中以便压缩堆栈。这种收集器也停止其他操作。 2.3.复制收集器 这种收集器将堆栈分为两个域，常称为半空间。每次仅使用一半的空间，jvm生成的新对象则放在另一半空间中。gc运行时，它把可到达对象复制到另一半空间，从而压缩了堆栈。这种方法适用于短生存期的对象，持续复制长生存期的对象则导致效率降低。 2.4.增量收集器 增量收集器把堆栈分为多个域，每次仅从一个域收集垃圾。这会造成较小的应用程序中断。 2.5.分代收集器 这种收集器把堆栈分为两个或多个域，用以存放不同寿命的对象。jvm生成的新对象一般放在其中的某个域中。过一段时间，继续存在的对象将获得使用期并转入更长寿命的域中。分代收集器对不同的域使用不同的算法以优化性能。 2.6.并发收集器 并发收集器与应用程序同时运行。这些收集器在某点上（比如压缩时）一般都不得不停止其他操作以完成特定的任务，但是因为其他应用程序可进行其他的后台操作，所以中断其他处理的实际时间大大降低。 2.7.并行收集器 并行收集器使用某种传统的算法并使

用多线程并行的执行它们的工作。在多cpu机器上使用多线程技术可以显着的提高java应用程序的可扩展性。 