Java理论与实践:垃圾收集简史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_E7_90_86_E8_AE_BA_c104_644966.htm 垃圾收集的好处是无可争辩的 —— 可靠性提高、使内存管理与类接口设计分离，并使开发者减少了跟踪内存管理错误的时间。着名的悬空指针和内存泄漏问题在 Java 程序中再也不会发生了（Java 程序可能会出现某种形式的内存泄漏，更精确地说是非故意的对象保留，但是这是一个不同的问题）。不过，垃圾收集不是没有代价的 —— 其中包括对性能的影响、暂停、配置复杂性和不确定的结束 (nondeterministic finalization)。 一个理想的垃圾收集实现应该是完全不可见的 —— 没有垃圾收集暂停、没有因为垃圾收集而产生的 CPU 时间损失、垃圾收集器不会与虚拟内存或者缓存有负面的互动，并且堆不需要大于应用程序的驻留空间（即堆占用）。当然，没有十全十美的垃圾收集器，但是垃圾收集器在过去十年中已经有了很大改进。 选项与选择 1.3 JDK 包括三种不同的垃圾收集策略，1.4.1 JDK 包括六种垃圾收集策略以及 12 种以上用于配置和优化垃圾收集的命令行选项。它们有什么不同？为什么需要有这么多选项？ 不同的垃圾收集实现使用不同的策略来识别和收回不可到达的对象，它们与用户程序和调度器以不同的方式互动。不同类型的应用程序对于垃圾收集有不同的要求 —— 实时应用程序会将要求收集暂停的持续时间短并且有限制，而企业应用程序可能允许更长时间和可预测性更低的暂停以获得更高的吞吐能力。 垃圾收集如何工作？ 有几种垃圾收集的基本策略：引用计数、标记-清除、标记-整理 (mark-compact) 和复制。此外

，一些算法可以以 增量方式完成它们的工作（不需要一次收集整个堆，使得收集暂停时间更短），一些算法可以在用户程序运行时运行（并发收集）。其他算法则必须在用户程序暂停时一次进行整个收集（即所谓的 stop-the-world收集器）。最后，还有混合型的收集器，如 1.2 和以后版本的 JDK 使用的分代收集器，它对堆的不同区域使用不同的收集算法。在对垃圾收集算法进行评价时，我们可能要考虑以下所有标准： 暂停时间。收集器是否停止所有工作来进行垃圾收集？要停止多长时间？暂停是否有时间限制？ 暂停的可预测性。垃圾收集暂停是否规划为在用户程序方便而不是垃圾收集器方便的时间发生？ CPU 占用。总的可用 CPU 时间用在垃圾收集上的百分比是多少？ 内存大小。许多垃圾收集算法需要将堆分割成独立的内存空间，其中一些空间在某些时刻对用户程序是不可访问的。这意味着堆的实际大小可能比用户程序的最大堆驻留空间要大几倍。 虚拟内存交互。在具有有限物理内存的系统上，一个完整的垃圾收集在垃圾收集过程中可能会错误地将非常驻页面放到内存中来进行检查。因为页面错误的成本很高，所以垃圾收集器正确管理引用的区域性 (locality) 是很必要的。 缓存交互。即使在整个堆可以放到主内存中的系统上 —— 实际上几乎所有 Java 应用程序都可以做到这一点，垃圾收集也常常会有将用户程序使用的数据冲出缓存的效果，从而影响用户程序的性能。 对程序区域性的影响。虽然一些人认为垃圾收集器的工作只是收回不可到达的内存，但是其他人认为垃圾收集器还应该尽量改进用户程序的引用区域性。整理收集器和复制收集器在收集过程中重新安排对象，这有可能改进区域性。 编译器和运行时影响。一

些垃圾收集算法要求编译器或者运行时环境的重要配合，如当进行指针分配时更新引用计数。这增加了编译器的工作，因为它必须生成这些簿记指令，同时增加了运行时环境的开销，因为它必须执行这些额外的指令。这些要求对性能有什么影响呢？它是否会干扰编译时优化呢？ 不管选择什么算法，硬件和软件的发展使垃圾收集更具有实用性。20 世纪 70 和 80 年代的经验研究表明，对于大型 Lisp 程序，垃圾收集消耗 25% 到 40% 的运行时。垃圾收集还不能做到完全不可见，这肯定还有很长的路要走。 