Report for FoPL

Stu:**金泽文**

No. PB15111604

摘要

通过阅读中村成洋的《垃圾回收的算法与实现》，理解经典垃圾回收算法的机制与应用。

目录

正文

一、 两类 GC：保守式 GC 与准确式 GC。

针对不同的使用环境与特定的实现要求，一般把 GC 分为两大类：保守式与准确式。

一般而言，在不能够精准地判断一个变量值是不是指针的前提下，只能采用保守式 GC。而准确式 GC 则只能在能判断的前提下使用。顾名思义，保守式会以可靠性为原则，将所有可能是指针的值全部按照指针处理，以免带来风

险。而准确式 GC 则没有此顾虑，可以实现更多大胆的想法，尤其是移动分块，这让准确式 GC 得到更好的内存利用效率。

GC 的选择与语言处理程序有很大的关系。因为指针和非指针的识别，一般需要借助特定的手段，才能够准确判断。对于 C 语言，union 数据类型会模糊数据的准确类型，随意的强制类型转换，也导致了判断的复杂性。所以这时只能采用保守 GC。而如果简单改变 c 的语法特性，在实现的过程中增加一些策略：比如对所有值增加限定，对根据指针的低两位为 0 的特性设置 tag 位，以辨别，同时删去 union 等不明确的数据结构等。除了对语法的修改，还可以在一定的限制条件下，通过一些策略将保守式 GC 改进为带有部分准确式 GC 功能的保守式 GC。比如 Joel F. Bartlett 提出的 MostlyCopingGC，打破了传统的保守式 GC 不能采用复制策略的壁障。再比如 Hans J. Boehm 发明的黑名单策略。

当然，为了实现准确式 GC 而带来的修改，肯定会增加性能上的降低。这一点不可避免，但是准确式 GC 也会带来垃圾的准确回收，在内存上得到更好的利用。

二、 几种经典 GC 算法

a) 标记 – 清除算法

这一算法，顾名思义，分为标记和清除两个阶段，可以说是最简单的 GC 算法。标记阶段将所有活动对象打上标记；清除阶段将所有未标记的对象回收。在分配空间的时候，一般有三种策略：First Fit, Best Fit, Worst Fit。First Fit 最为简单，Worst Fit 效果最糟糕。First Fit 和 Best Fit 两种策略在不同的场合下有不同的表现，视情况选择。

标记 – 清除算法的最大优点就是算法简单，最容易实现；而缺点也很明显，那就是没有任何的移动，会导致大量细化的碎片产生，会严重影响内存的使用率。一般选择这一算法时会结合其他算法进行混合使用。

而针对碎片化以及的问题，有一些改进的变种。

比如 BiBOP，通过将大小相近的对象整理成固定大小的块进行管理，这样能在一定程度上减缓碎片话的问题，但是效果很小。

同时，标记 – 清除算法还有一个问题，那就是持续的写入标记位会带来另一个影响：那就是对写时复制技术的阻碍。比如 linux 下的 fork 函数，生成的进程可能和原进程十分相近，那么这是如果复制两份，那么内存的使用率就会不划算，所以写时复制技术的提出有力的提高了内存的效率。而如果持续写如标记，那么原来的写时复制带来的优势也荡然无存。

针对这一问题，有人提出了位图表格，简单来说就是把标记位统一放在内存的一个地方，而不占用原进程的内存，所以也就不用持续写入原进程的内存了。

b) 引用计数法

一种自然的标记想法是利用计数器的概念，这就得到来了引用计数法。C++的 smart pointer 就引入了这一概念。

相对于标记 – 清除算法，引用计数法可能要繁琐一些：每当生成一个新指针时计数器加一，每当改变一个指针时，后来的加一原来的减一。由于这个减一的操作，在每次减一的时候就可以检查一下计数器是否为 0，如果是 0，即可立即收回。这样带来的一大好处就是最大暂停时间的大大缩

短，同时，也不用像标记-清除法一样每次沿着指针遍历。这些都是引用计数法的优点。当然，它也有缺点，比如计数器占用的位数比标签占用的多，每次加一减一也需要更多的操作。此消彼长，具体的效率性能还是要根据实际场合判断。

关于引用计数法永远不能不谈的一点，就是循环引用的问题。

本来在 project 中我想用引用计数法，想用 c++的 smart pointer 来处理 gc 问题，但是循环引用的问题实在是不能忽视。因为我的 interpreter 是建立在原生的语法树上的，所以，由于 father 和 son 之间的互相引用，所以很难使用 smart pointer。虽然也有很多针对循环引用问题的补救策略，但是效果都不是那么尽人意，比如部分标记 - 清除算法。

部分标记 – 清除算法通过将对象染成四种颜色：黑白灰阴影，采用一定搜索策略之后，锁定可能是垃圾的循环引用的对象，来达到准确回收的目的。然而，部分标记-清除算法的局限性也很明显，那就是操作实在太多，将最大暂停时间大大增长。而 c++针对循环引用的 smart pointer 问题，在 c++11 中增加了 weak_ptr 来避免。

c) GC 复制算法

之前提到的碎片化的问题，必须得到解决，而将一块对象移动到另一块则是很自然的想法，于是 Marvin L. Minsky 提出了 GC 复制算法。

这一算法通过将空间分为两个部分，一个活动部分，一个保留部分，在需要的空间不足时来回赋值转移来达到是对象之间的分配更加紧凑的目的。所以优点很明显：碎片的消除；缺点更加显著：空间利用率永远不足 50%。

一种变种是多空间复制算法，将通过将一部分空间分配给标记-清除算法，另一部分使用复制算法，来减少空间的空闲率。

d) 标记-压缩算法

顾名思义，标记-压缩算法由标记、压缩两个阶段组成。典型的有 Donald E. Knuth 提出的 Lisp2 算法，和 Robert A. Saunders 提出的 Two-finger 压缩算法。

两者都是用压缩的方法充分利用了内存，避免了碎片化。后者可以说只是前者在某些条件下的优化。

e） 分代 GC

分代 GC 中引入了年龄的概念，这一点与多级优先调度算法有异曲同工之妙，都是通过存在的时间来判断。只不过，分代 GC 里面，“年龄”越大的对象，越不会回收。

分代 GC 的设计思路只要是来源于这样一个经验——“往往是年轻的局部变量死的早”。于是就有了分代 GC。分代 GC 的优点是结合实际经验，吞吐量的到改善；但是缺点则是实际经验偶尔会犯错，倒是跟预期不同的结果。

总结

理论必须联系实际。

目前，不同的编程语言根据自己不同的需要采用者不同的 GC 策略。

对于 python 来说，它相对地更注重实时性，所以它使用的主要是最大暂停时间最短的引用计数法，设计了 3 个独立的分配空间的层次。由低到高依次负责地基内存，对象，以及对象特有的内存。针对循环引用的问题，Cpython

设计了特别的模块——GC module，来检查循环引用的问题。而这一模块的实现，引入了标记 – 清除算法和分代 GC 技术。另外值得一提的是，为了性能，python 还特地设计了大量的内存池机制。

Google Chrome 是目前市场占有率不错的浏览器，并且以它的 V8 引擎而闻名。V8 是 Google 独立开发的 JavaScript 引擎，实现了准确式 GC，整体上采用了分代垃圾回收机制。针对新生代 GC，采用 cheney 的 GC 复制算法；针对老年代 GC，采用的是标记-清除算法和标记-压缩算法。另外，值得一提的是，V8 的高速运行的来源之一，在于它采用的 JIT 技术，不同于传统的解释技术，直接采用即时编译的方法得到极高的速度。

GC 的各种算法，在没有全新的硬件进化之前，很难有质的飞跃。多数情况是针对某一种新的语言处理要求，多方面糅合经典 GC，针对具体的需求进行部署。