精品文章 | [译文] from ruby-china

Ruby与Python的GC

前言

本文基于我在刚刚过去的在布达佩斯举行的RuPy上的演讲。我觉得趁热打铁写成帖子应该会比只留在幻灯片上更有意义。你也可以看看演讲录像。再跟你说件事，我在Ruby大会也会做一个相似的演讲，但是我不会去说Python的事儿，相反我会对比一下MRI,JRuby和Rubinius的垃圾回收机制。

想了解Ruby垃圾回收机制和Ruby内部实现更详尽的阐述，请关注即将问世的拙作《Ruby Under a Microscope》。

既然是”Ruby Python”大会，我觉得对比一下Ruby和Python的垃圾回收机制应该会很有趣。在此之前，到底为什么要计较垃圾回收呢？毕竟，这不是什么光鲜亮丽激动人心的主题，对吧。你们大家有多少人对垃圾回收感冒？(竟然有不少RuPyde与会者举手了!)

最近Ruby社区发表了一篇博文,是关于如何通过更改Ruby GC设置来为单元测试提速的。我认为这篇文章是极好的。对于想让单元测试跑得更快和让程序GC暂停更少的人来说很有裨益，但是GC并没能引起我的兴趣。第一瞥GC就像是一个让人昏昏欲睡的、干巴巴的技术主题。

但是实际上垃圾回收是一个迷人的主题：GC算法不仅是计算机科学史的重要组成部分，也是一个前沿课题。举例来说，MRI Ruby使用的标记-清除算法已经年逾五

旬了，而Ruby的替代语言Rubinius使用的GC算法在不久前的2008年才被发明出来。

然而，”垃圾回收”这个词其实有些用词不当。

应用程序那颗跃动的心

GC系统所承担的工作远比”垃圾回收”多得多。实际上，它们负责三个重要任务。它们

为新生成的对象分配内存
识别那些垃圾对象，并且
从垃圾对象那回收内存。
如果将应用程序比作人的身体：所有你所写的那些优雅的代码，业务逻辑，算法，应该就是大脑。以此类推，垃圾回收机制应该是那个身体器官呢？（我从RuPy听众那听到了不少有趣的答案：腰子、白血球 :)）

我认为垃圾回收就是应用程序那颗跃动的心。像心脏为身体其他器官提供血液和营养物那样，垃圾回收器为你的应该程序提供内存和对象。如果心脏停跳，过不了几秒钟人就完了。如果垃圾回收器停止工作或运行迟缓,像动脉阻塞,你的应用程序效率也会下降，直至最终死掉。

```python
class Node:
  def __init__(self, val):
    self.value = val

print(Node(1))
print(Node(2))
```

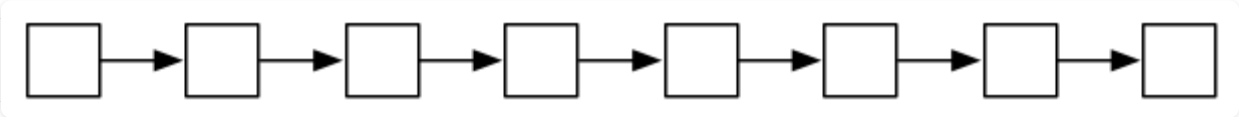
```ruby
class Node
  def initialize(val)
    @value = val
  end
end

p Node.new(1)
p Node.new(2)
```

顺便提一句，两种语言的代码竟能如此相像：Ruby 和 Python 在表达同一事物上真的只是略有不同。但是在这两种语言的内部实现上是否也如此相似呢?

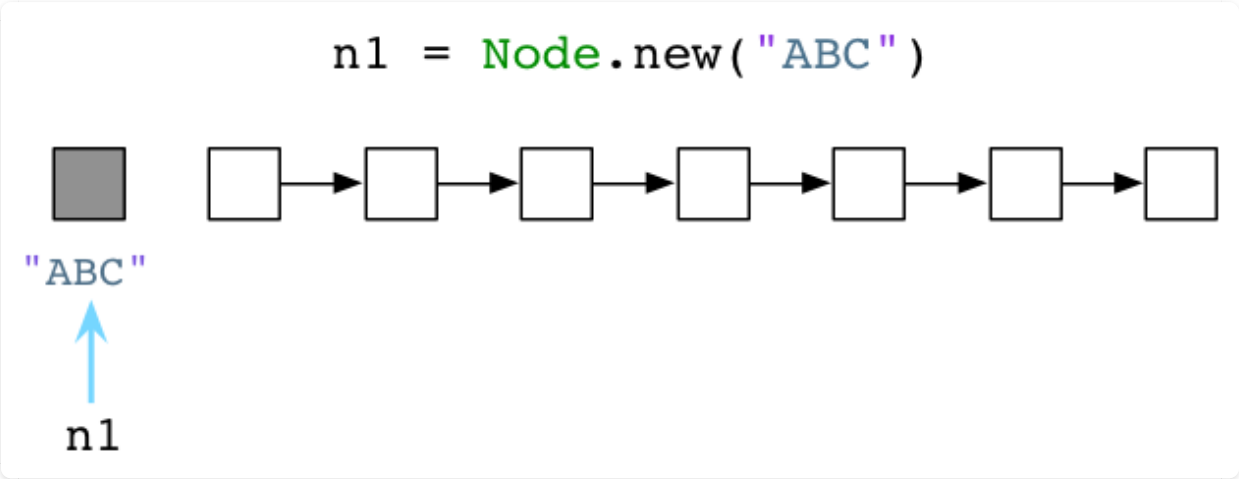
可用列表

当我们执行上面的Node.new(1)时，Ruby到底做了什么？Ruby是如何为我们创建新的对象的呢?

出乎意料的是它做的非常少。实际上，早在代码开始执行前，Ruby就提前创建了成百上千个对象，并把它们串在链表上，名曰：可用列表。下图所示为可用列表的概念图：

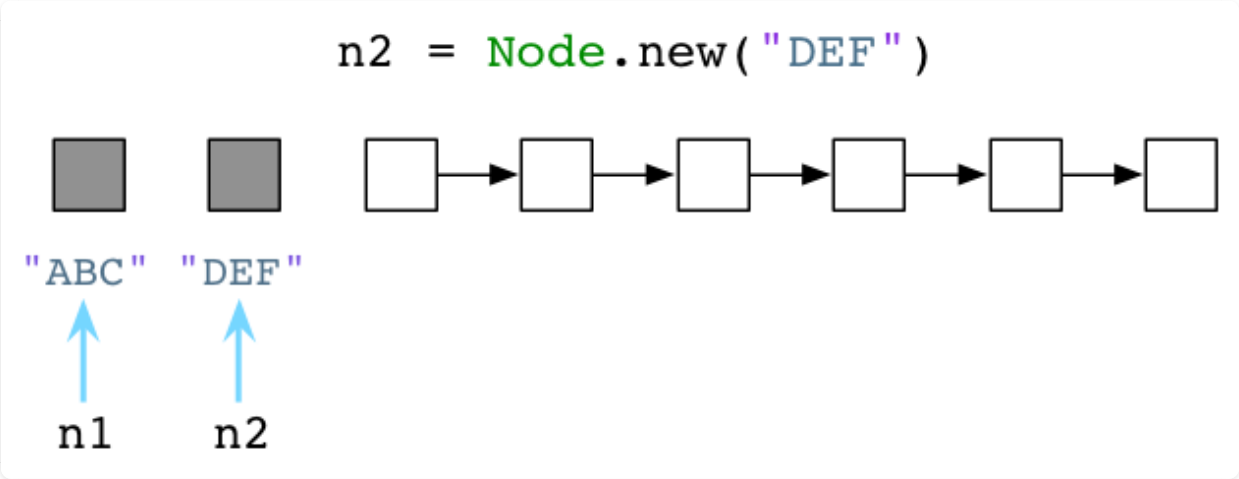


想象一下每个白色方格上都标着一个”未使用预创建对象”。当我们调用 Node.new ,Ruby只需取一个预创建对象给我们使用即可：



上图中左侧灰格表示我们代码中使用的当前对象，同时其他白格是未使用对象。(请注意：无疑我的示意图是对实际的简化。实际上，Ruby会用另一个对象来装载字符串”ABC”,另一个对象装载Node类定义，还有一个对象装载了代码中分析出的抽象语法树，等等)

如果我们再次调用 Node.new，Ruby将递给我们另一个对象：



这个简单的用链表来预分配对象的算法已经发明了超过50年，而发明人这是赫赫有

名的计算机科学家John McCarthy，一开始是用Lisp实现的。Lisp不仅是最早的函数式编程语言，在计算机科学领域也有许多创举。其一就是利用垃圾回收机制自动化进行程序内存管理的概念。

标准版的Ruby，也就是众所周知的”Matz’s Ruby Interpreter”(MRI),所使用的GC算法与McCarthy在1960年的实现方式很类似。无论好坏，Ruby的垃圾回收机制已经53岁高龄了。像Lisp一样，Ruby预先创建一些对象，然后在你分配新对象或者变量的时候供你使用。

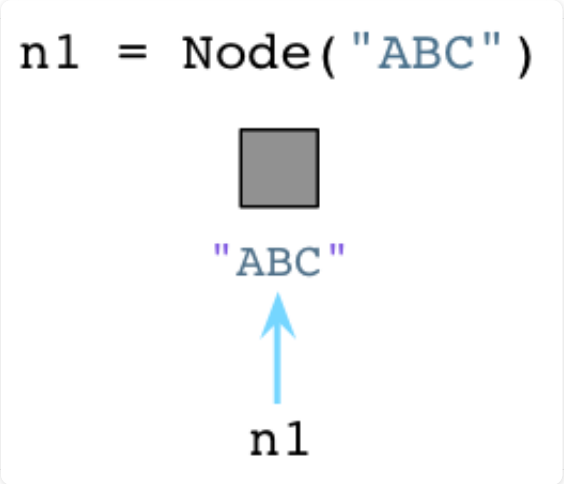
> *金勇注: Ruby机制: 包工作分配.*

Python的对象分配

我们已经了解了Ruby预先创建对象并将它们存放在可用列表中。那Python又怎么样呢?

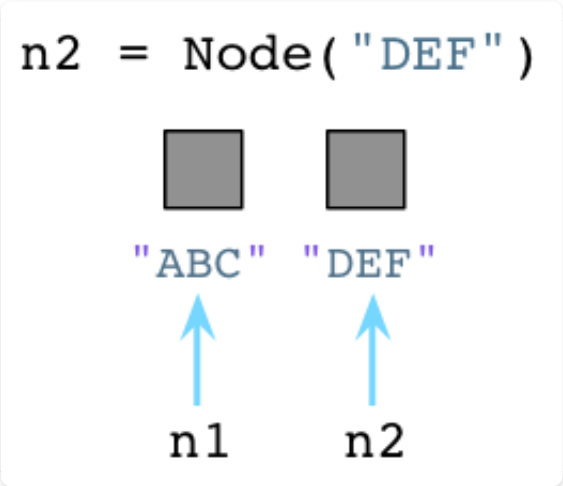
尽管由于许多原因Python也使用可用列表(用来回收一些特定对象比如 list)，但在为新对象和变量分配内存的方面Python和Ruby是不同的。

例如我们用Pyhon来创建一个Node对象：



与Ruby不同，当创建对象时Python立即向操作系统请求内存。(Python实际上实现了一套自己的内存分配系统，在操作系统堆之上提供了一个抽象层。但是我今天不展开说了。)

当我们创建第二个对象的时候，再次像OS请求内存：

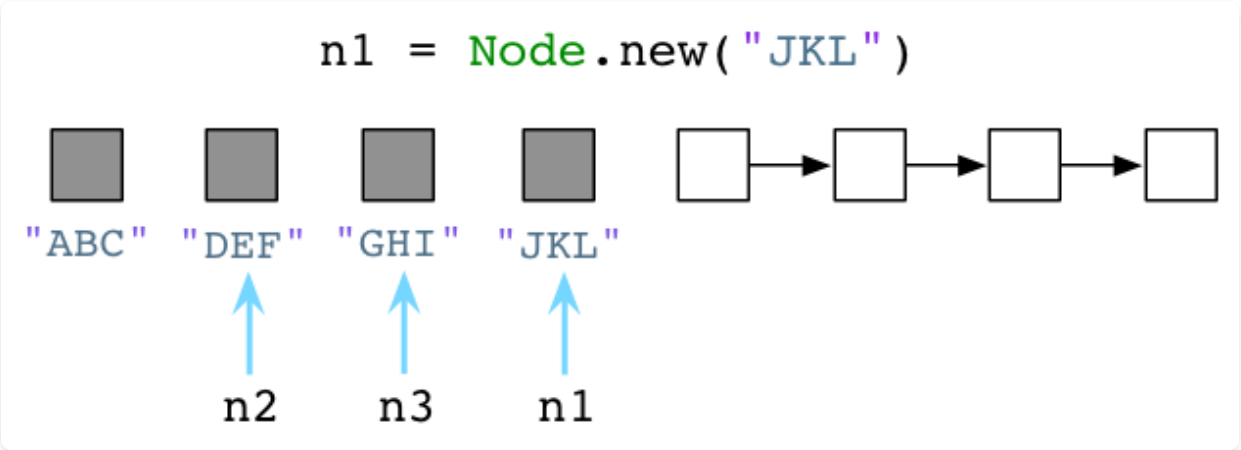


看起来够简单吧，在我们创建对象的时候，Python会**花些时间**为我们找到并分配内存。

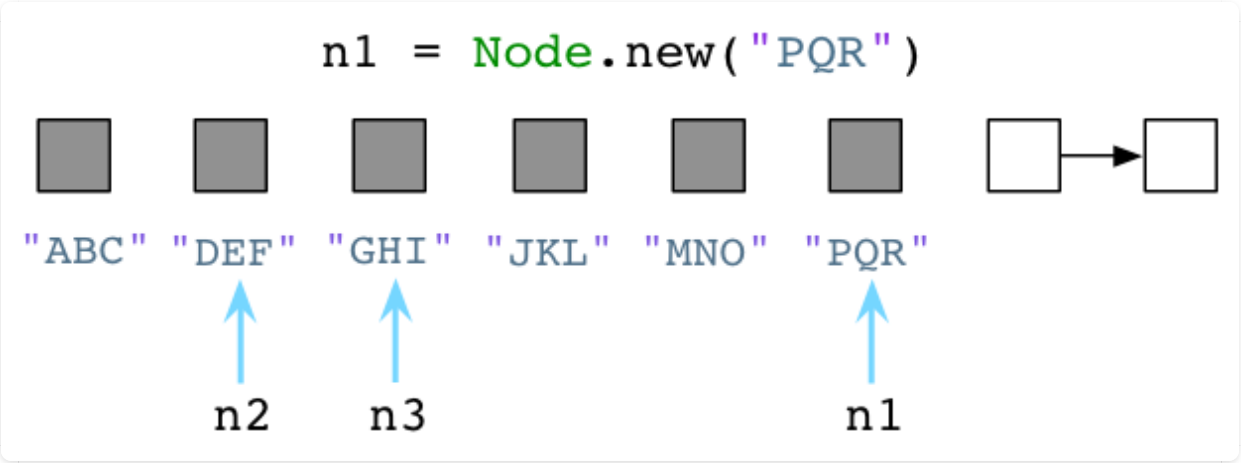
> *金勇注: Python机制, 自己寻出路.*

Ruby 开发者住在凌乱的房间里

回过来看Ruby。随着我们创建越来越多的对象，Ruby会持续寻可用列表里取预创建对象给我们。因此，可用列表会逐渐变短:



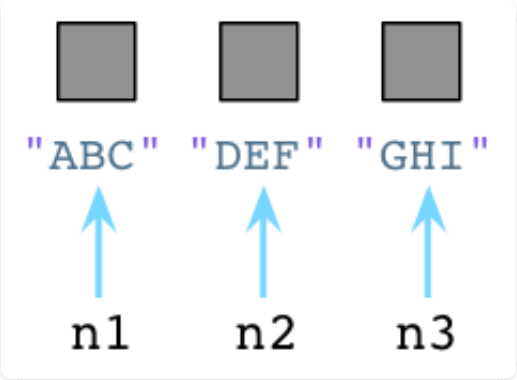
...然后更短：



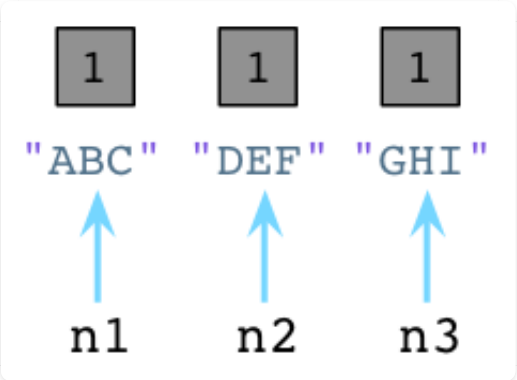
请注意我一直在为变量n1赋新值，Ruby把旧值留在原处。”ABC”,”JKL”和”MNO”三个Node实例还滞留在内存中。Ruby不会立即清除代码中不再使用的旧对象！Ruby开发者们就像是住在一间凌乱的房间，地板上摞着衣服，要么洗碗池里都是脏盘子。作为一个Ruby程序员，无用的垃圾对象会一直环绕着你。

Python 开发者住在卫生之家庭

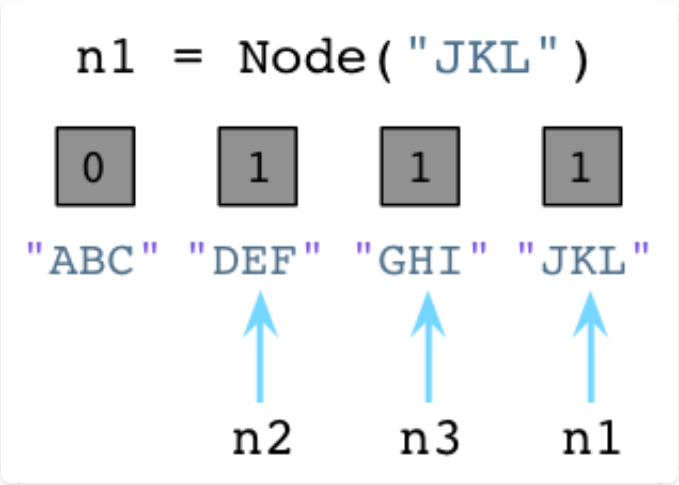
Python与Ruby的垃圾回收机制颇为不同。让我们回到前面提到的三个Python Node对象：



在内部，创建一个对象时，Python总是在对象的C结构体里保存一个整数，称为引用数。期初，Python将这个值设置为1：

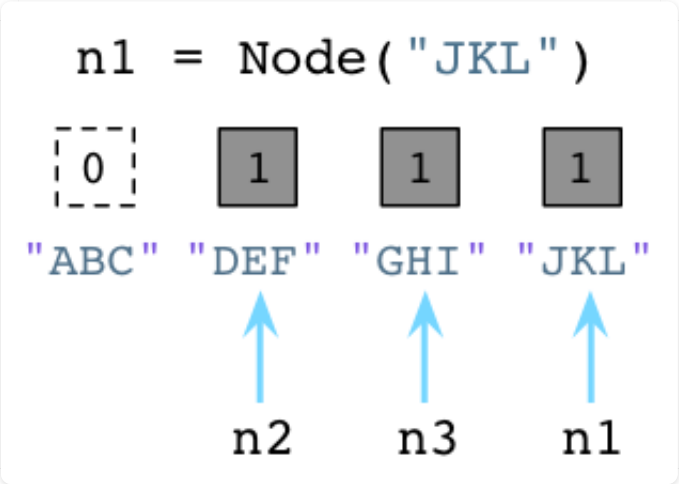


值为1说明分别有个一个指针指向或是引用这三个对象。假如我们现在创建一个新的Node实例，JKL：



与之前一样，Python设置JKL的引用数为1。然而，请注意由于我们改变了n1指向了JKL，不再指向ABC，Python就把ABC的引用数置为0了。

此刻，Python垃圾回收器立刻挺身而出！每当对象的引用数减为0，Python立即将其释放，把内存还给操作系统：

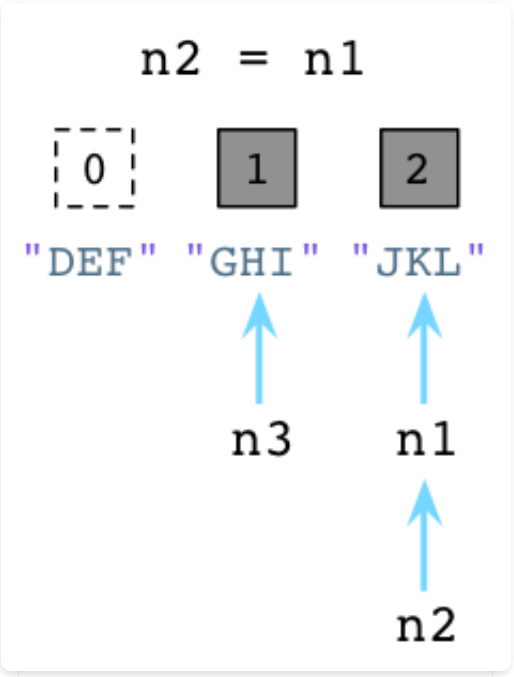


上面Python回收了ABC Node实例使用的内存。记住，Ruby弃旧对象原地于不顾，也不释放它们的内存。

Python的这种垃圾回收算法被称为引用计数。是George Collins在1960年发明的，恰巧与John McCarthy发明的可用列表算法在同一年出现。就像Mike Bernstein在6月份哥谭市Ruby大会杰出的垃圾回收机制演讲中说的:"1960年是垃圾收集器的黄金年代..."

Python开发者工作在卫生之家，你可以想象，有个患有轻度[OCD][]的室友一刻不停地跟在你身后打扫，你一放下脏碟子或杯子，有个家伙已经准备好把它放进洗碗机了!

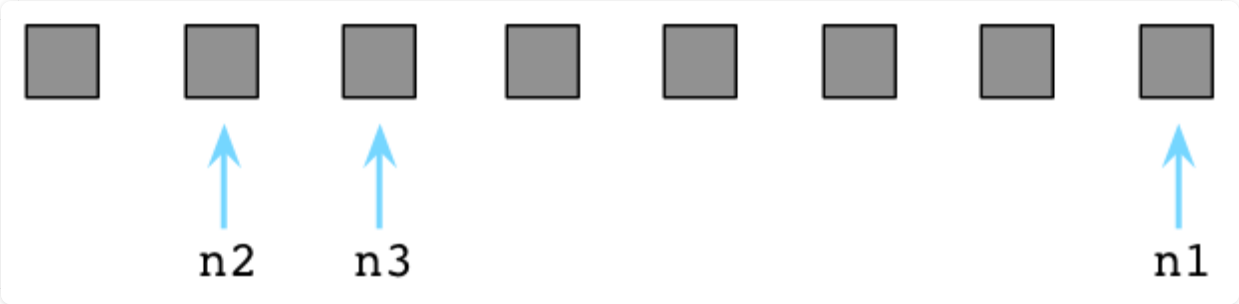
现在来看第二例子。加入我们让n2引用n1:



上图中左边的DEF的引用数已经被Python减少了，垃圾回收器会立即回收DEF实例。同时JKL的引用数已经变为了2 ，因为n1和n2都指向它。

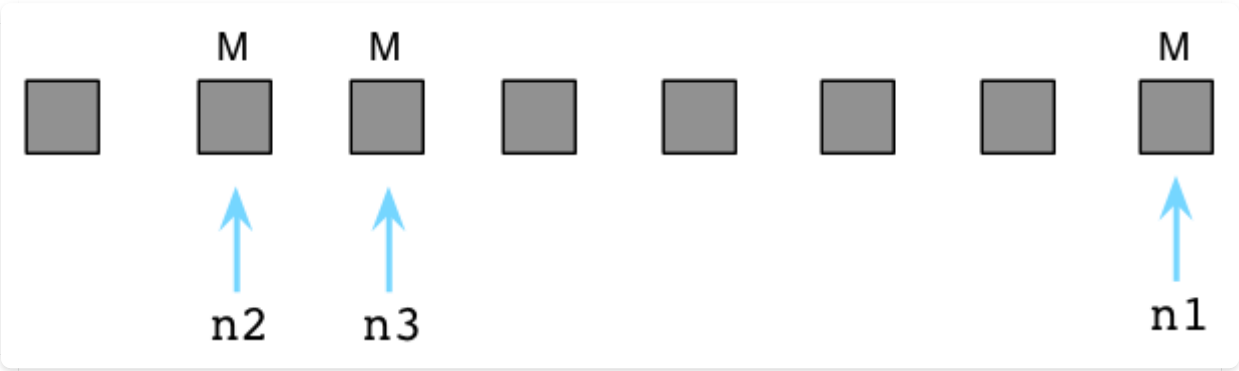
标记-清除

最终那间凌乱的房间充斥着垃圾，再不能岁月静好了。在Ruby程序运行了一阵子以后，可用列表最终被用光光了:

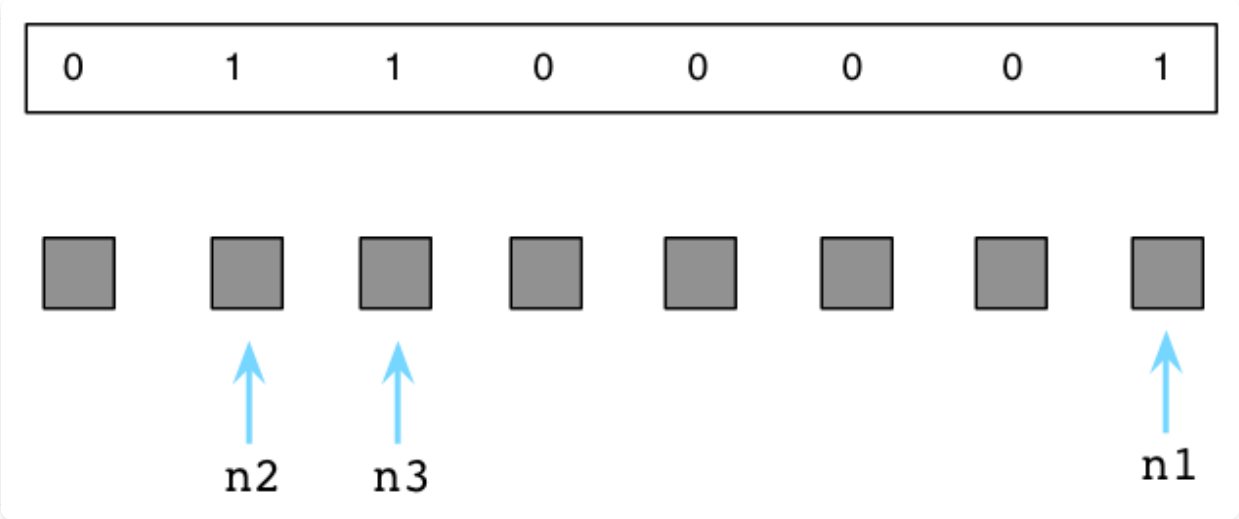


此刻所有Ruby预创建对象都被程序用过了(它们都变灰了)，可用列表里空空如也（没有白格子了）。

此刻Ruby祭出另一McCarthy发明的算法，名曰：标记–清除。首先Ruby把程序停下来，Ruby用”地球停转垃圾回收大法”。之后Ruby轮询所有指针，变量和代码产生别的引用对象和其他值。同时Ruby通过自身的虚拟机便利内部指针。标记出这些指针引用的每个对象。
我在图中使用M表示。

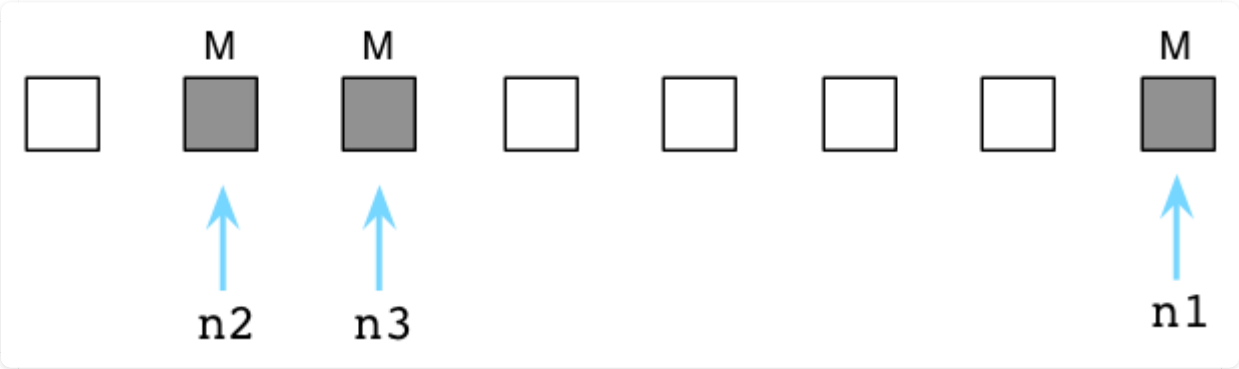


上图中那三个被标M的对象是程序还在使用的。在内部，Ruby实际上使用一串位值，被称为:可用位图(译注：还记得《编程珠玑》里的为突发排序吗，这对离散度不高的有限整数集合具有很强的压缩效果，用以节约机器的资源。)，来跟踪对象是否被标记了。



Ruby将这个可用位图存放在独立的内存区域中，以便充分利用Unix的写时拷贝化。

如果说被标记的对象是存活的，剩下的未被标记的对象只能是垃圾，这意味着我们的代码不再会使用它了。我会在下图中用白格子表示垃圾对象：



接下来Ruby清除这些无用的垃圾对象，把它们送回到可用列表中.

在内部这一切发生得迅雷不及掩耳，因为Ruby实际上不会吧对象从这拷贝到那。而是通过调整内部指针，将其指向一个新链表的方式，来将垃圾对象归位到可用列表中的。

现在等到下回再创建对象的时候Ruby又可以把这些垃圾对象分给我们使用了。在Ruby里，对象们六道轮回，转世投胎，享受多次人生。

标记-删除 vs. 引用计数

乍一看，Python的GC算法貌似远胜于Ruby的：宁舍洁宇而居秽室乎？为什么Ruby宁愿定期强制程序停止运行，也不使用Python的算法呢?

然而，引用计数并不像第一眼看上去那样简单。有许多原因使得不许多语言不像

Python这样使用引用计数GC算法：

首先，它不好实现。Python不得不在每个对象内部留一些空间来处理引用数。这样付出了一小点儿空间上的代价。但更糟糕的是，每个简单的操作（像修改变量或引用）都会变成一个更复杂的操作，因为Python需要增加一个计数，减少另一个，还可能释放对象。
第二点，它相对较慢。虽然Python随着程序执行GC很稳健（一把脏碟子放在洗碗盆里就开始洗啦），但这并不一定更快。Python不停地更新着众多引用数值。特别是当你不再使用一个大数据结构的时候，比如一个包含很多元素的列表，Python可能必须一次性释放大量对象。减少引用数就成了一项复杂的递归过程了。
最后，它不是总奏效的。在我的下一篇包含了我这个演讲剩余部分笔记的文章中，我们会看到，引用计数不能处理环形数据结构--也就是含有循环引用的数据结构。

补充

下周我会分解演讲的剩余部分。我会讨论一下Python如何摆平环形数据类型及GC在即将出炉的Ruby2.1发行版中是如何工作的。

[OCD]: Obsessive compulsive disorder 强迫症即强迫性神经症，亦译沉溺，是一种神经官能症，为焦虑症的一种。患有此病的患者总是被一种入侵式的思维所困扰，在生活中反复出现强迫观念及强迫行为，使到患者感到不安、恐慌或者担忧等等，从而进行某种重复行为，至使舒缓其此种压迫感受。患者自知力完好，对于症状了解，然而无法摆脱强迫行为。 强迫症是世界上最常见精神问题中的第四位，其病发率跟哮喘及糖尿病同样普遍。在美国，每50个人就有一人可能是强迫症患者。