增量式垃圾回收

原创 一蓑烟雨17815 2015-12-27 23:24:02 2089 收藏 1 版权

分类专栏： 虚拟机技术 文章标签： 编译原理 数据结构 GC 垃圾回收

简单的增量式垃圾回收

通过这段时间对tinypy源码和编译原理这本书的研究，我终于敲开了增量式垃圾回收的“小门”；如果读者没有接触过基本的标记-清扫垃圾回收，最好先对其进行一定的了解。

一、数据结构

列表：

- 所有对象集合，标记为R，
- 未扫描对象集合，标记为U，
- 垃圾对象集合，标记为G

二、算法的理论基础

- 不可达对象永远不会变成可达对象。
- 树的遍历（增量标记的核心）

三、算法的执行过程

初始化

- 初始化栈和对象的树；
- 将根对象push进U；

增量标记

（实质上是对可达对象的树进行广度优先的遍历过程，这个过程中会产生一些被称之为“漂浮垃圾”的对象，他们已经不可达，但是仍然在U集合里面，由于他们不可能再次被引用，所以在下一轮FullGC过程中会被清除）

- 从U中pop出一个对象
- 如果该对象已经标记为1，退出过程
- 如果该对象有子节点，将子节点push进U
- 标记该对象为1
- （这个过程也可以增量清除G中的垃圾对象）

生成新对象

- 标记该对象为1
- 将新对象push进U
- 执行两步增量标记，保证U不断减小
- 如果U为空，则表明所有对象都被标记过了，R中所有标记为1的为垃圾对象，遍历出这些对象，加入G，（当然，可以立即释放这些对象，但是这需要Stop The World）

访问对象

有两种方式

- 拦截对象读操作
- 拦截对象写操作

由于大部分程序写操作少于读操作，考虑到性能，通常我们拦截写操作

- 将写操作中引用关系可能发生变化的对象push进U
- 执行增量标记（可以多步）

```
// 比如下面操作
obj.name = value // dict是一个对象，string是一个字符串
// 这里修改了dict对象的name属性
// 那么需要进行标记的就是obj和value两个对象
// 因为obj和value有可能之前没有引用关系
```

全量GC

- 释放R中标记为0的对象；
- 将R中所有对象标记为0；
- 将跟对象push进U

【注意】可以根据需要调整增量跟踪的频率

四、增量垃圾收集的可行性分析

1. 第一次gc过程中不会产生任何垃圾，因为新对象和可达对象都被标记；
2. 第二次gc过程中收集第一次gc之前产生的垃圾，因为所有对象的标记都被重置为0，而不可达对象不可再变成可达，因此第一次gc中的垃圾会保持未标记状态；
3. 依次类推，垃圾总能在下一次gc过程中被完全回收；

五、出现的问题以及思考

在此之前，我考虑过增变过程是否需要跟踪改变的对象，因为如果引用新对象，那么该对象已在产生的时候标记并且放入已扫描列表，如果引用旧对象，那么旧对象要么已经标记，要么将在后面标记，这样听起来似乎有些道理。

事实上，这个推论有一个错误，试想这样一个情况

```
var b = new Person(); // b被标记为1
a.friend = b; //
a.friend = c; //
// 发生全量GC，b被标记为0
d.friend = b; // 再次引用b，但是变更过程没被跟踪
// 全量GC，b被回收
```

因此，拦截所有的写操作是必须的。