< 2018年8月 >

日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

公告

昵称：凌动小生
园龄：6年1个月
粉丝：4
关注：0
+加关注

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

毕业季(1)
随笔(1)

随笔分类

JVM--标记-清除算法Mark-Sweep

前言

垃圾自动回收机制的出现使编程更加的简单，使得我们不需要再去考虑内存分配和释放的问题，而是更加的专注在我们产品功能的实现上。但是我们还是需要花时间去了解下垃圾收集机制是怎么工作的，以便后面能够更好的进行我们应用的性能调优等。

目前最基本的垃圾收集算法有四种,标记-清除算法(mark-sweep),标记-压缩算法(mark-compact),复制算法(copying)以及引用计数算法(reference counting).而现代流行的垃圾收集算法一般是由这四种中的其中几种算法相互组合而成，比如说，对堆(heap)的一部分采用标记-清除算法，对堆(heap)的另外一部分则采用复制算法等等。今天我们主要来看下标记-清除算法的原理。

基本概念

在了解标记-清除算法前，我们先要了解几个基本概念。

首先是mutator和collector，这两个名词经常在垃圾收集算法中出现，collector指的就是垃圾收集器，而mutator是指除了垃圾收集器之外的部分，比如说我们应用程序本身。mutator的职责一般是NEW(分配内存),READ(从内存中读取内容),WRITE(将内容写入内存)，而collector则就是回收不再使用的内存来供mutator进行NEW操作的使用。

第二个基本概念是关于mutator roots(mutator根对象),mutator根对象一般指的是分配在堆内存之外，可以直接被mutator直接访问到的对象，一般是指静态/全局变量以及Thread-Local变量(在Java中，存储在java.lang.ThreadLocal中的变量和分配在栈上的变量 - 方法内部的临时变量等都属于此类).

第三个基本概念是关于可达对象的定义，从mutator根对象开始进行遍历，可以被访问到的对象都称为是可达对象。这些对象也是mutator(你的应用程序)正在使用的对象。

算法原理

顾名思义，标记-清除算法分为两个阶段，标记(mark)和清除(sweep).

在标记阶段，collector从mutator根对象开始进行遍历，对从mutator根对象可以访问到的对象都打上一个标识，一般是在对象的header中，将其记录为可达对象。

而在清除阶段，collector对堆内存(heap memory)从头到尾进行线性的遍历，如果发现某个对象没有标记为可达对象-通过读取对象的header信息，则就将其回收。

android(5)
database(2)
FreeBSD(6)
iPhone dev(17)
java (37)
javascript(9)
linux(2)
sencha touch(3)
ssh(4)
计算机专业(5)
日志-随笔(1)
心理拾掇(2)

随笔档案

2016年11月 (1)
2016年6月 (2)
2016年1月 (1)
2015年11月 (2)
2015年10月 (3)
2015年9月 (1)
2015年8月 (2)
2015年7月 (1)
2015年3月 (2)
2015年1月 (9)
2014年12月 (3)
2014年6月 (1)
2014年5月 (2)
2014年1月 (2)
2013年12月 (4)
2013年11月 (1)
2013年10月 (3)
2013年9月 (15)
2013年8月 (2)
2013年6月 (1)
2013年5月 (1)
2013年4月 (6)
2013年3月 (10)
2013年2月 (1)
2013年1月 (1)
2012年8月 (2)
2012年7月 (21)

最新评论

1. Re:人人网JavaScript面试题
免费域名无法访问了。

从上图我们可以看到，在Mark阶段，从根对象1可以访问到B对象，从B对象又可以访问到E对象，所以B,E对象都是可达的。同理，F,G,J,K也都是可达对象。到了Sweep阶段，所有非可达对象都会被collector回收。同时，**Collector在进行标记和清除阶段时会将整个应用程序暂停(mutator)，等待标记清除结束后才会恢复应用程序的运行，这也是Stop-The-World这个单词的来历。**

接着我们先看下一般垃圾收集动作是怎么被触发的，下面是mutator进行NEW操作的伪代码：

```
New():
        ref  <-  allocate()    //分配新的内存到ref指针
        if  ref  ==  null
              collect()    //内存不足，则触发垃圾收集
              ref  <-  allocate()
              if  ref  ==  null
                    throw  "Out  of  Memory"      //垃圾收集后仍然内存不足，则抛出Out  of  Memory错误
                    return  ref

atomic  collect():
        markFromRoots()
        sweep(HeapStart,HeapEnd)
```

而下面是对应的mark算法：

```
markFromRoots():
        worklist  <-  empty
        for  each  fld  in  Roots    //遍历所有mutator根对象
                ref  <-  *fld
                if  ref  !=  null  &&  isNotMarked(ref)    //如果它是可达的而且没有被标记的，直接标记该对象并将其加到
worklist中
                      setMarked(ref)
                      add(worklist,ref)
                      mark()
mark():
        while  not  isEmpty(worklist)
                      ref  <-  remove(worklist)    //将worklist的最后一个元素弹出，赋值给ref
                      for  each  fld  in  Pointers(ref)    //遍历ref对象的所有指针域，如果其指针域(child)是可达的，直接标记其
为可达对象并且将其加入worklist中
                      //通过这样的方式来实现深度遍历，直到将该对象下面所有可以访问到的对象都标记为可达对象。
                                  child  <-  *fld
                                  if  child  !=  null  &&  isNotMarked(child)
```

--快乐八哥


```
        setMarked(child)
        add(worklist,child)
```

在mark阶段结束后，sweep算法就比较简单了，它就是从堆内存起始位置开始，线性遍历所有对象直到堆内存末尾，如果该对象是可达对象的（在mark阶段被标记过的），那就直接去除标记位（为下一次的mark做准备），如果该对象是不可达的，直接释放内存。

```
sweep(start,end):
        scan  <-  start
      while  scan  <  end
             if  isMarked(scan)
                   setUnMarked(scan)
           else
                   free(scan)
           scan  <-  nextObject(scan)
```

缺点

标记-清除算法的比较大的缺点就是垃圾收集后有可能会造成大量的内存碎片，像上面的图片所示，垃圾收集后内存中存在三个内存碎片，假设一个方格代表1个单位的内存，如果有一个对象需要占用3个内存单位的话，那么就会导致Mutator一直处于暂停状态，而Collector一直在尝试进行垃圾收集，直到Out of Memory。

分类: java

« 上一篇：JVM 垃圾回收器工作原理及使用实例介绍
» 下一篇：转：面试题：“你能不能谈谈，java GC是在什么时候，对什么东西，做了什么事情？”

posted on 2015-10-28 16:41 凌动小生 阅读(1186) 评论(0) 编辑 收藏