关于jvm中卡表和写屏障的讨论

问题描述

为了深入了解GC运行机制，我正在阅读一些相关材料。偶然发现叫做“card table”的字眼，我Google它，但是没有找到让我信服的信息。绝大多数的解释很肤浅模糊。

我的问题：

- 卡表和写屏障如何工作？
- 卡表里面记录了什么？
- 垃圾收集器怎么知道特定的对象被其他下一代中存活对象引用？

我已经迫不及待，想得到一些关于这些问题实际性的知识。

回答1

我不确定你是不是看到了巨low的描述还是想知道更多细节，我很满意已经看到的解释。如果描述得很简练，辣么因为它的机制真的简单。

你显然已经知道，分代垃圾收集器需要扫描被老对象引用的新对象，扫描所有的老对象看起来很合适，但是破坏了某些分代的优势，所以你需要慢慢淡忘它。不管怎样，你需要写屏障-每当一些变量(引用数据类型)被写入或者被分配。如果一个新的引用指向一个新生代对象而且该引用存储在老年代对象中，写屏障为了垃圾收集而进行记录。关键在于它如何记录，标准方案是使用被叫做 remembered sets，一个存储所有存放一个(或者数个)指向新生代对象引用的老年代对象。你可以想象一下，它确实一点空间。

卡表是一种取舍： 告诉你一对象包含指向新生带对象引用(或者至少在某一时刻已经操作完成)，它将对象分组存到一个固定尺寸的区域并且可索引到这些老年对象。这样当然降低了空间使用率。但是非常正确是，你不需要关心如何存储这些对象，只要始终关注卡表。为了效率，你可以根据内存地址将这些老年代对象组织起来(因为可以很方便获取到)，花大力气将两者区分开来（按位操作去区分）。

而且不用维护一个区域列表，你也不用预先分配一个为了可能出现的区域而准备的空间。特别指出，它是一个N字节的字节数组，N是容器的数量，所以如果第i容器没有包含新生代对象，辣么第i个元素的值是0。 卡表用在这中情况下很合适。有代表性的是这个空间的分配，而且由堆释放。如果它不需要扩展的话，甚至被内置在内存的最初时期。除非全部空间被划分为堆(非常罕见)，超过了根据start_of_memory_region >> K公式计算得出数字代替0。所以为了根据idnex进入卡表，你必须减掉堆开始的内存地址。

总之，写屏障发现了“some_obj.field = other_obj;”类似的声明，在老年代对象里面存储一个新生代对象的引用，它是这样做的：

```
card_table[(&old_obj - start_of_heap) >> K] = 1;
```

上面的 &old_obj是现在拥有新生代对象指针老年对象的内存地址(当被判定指向新生对象的引用的时候已经进行注册)。minor GC 期间，垃圾收集器查看卡表为依据，扫描因为拥有新生代对象指针的堆区：

```
for i from 0 to (heap_size >> K):
    if card_table[i]:
        scan heap[i << K .. (i + 1) << K] for young pointers
```

回答2

前一段时间我写一篇关于解释HotSpot JVM新生代垃圾收集机制的文章Understanding GC pauses in JVM, HotSpot's minor GC。

> 写屏障脏卡的原理非常简单，每当程序在内存中修改引用的时候，将修改的内存页设置为脏，JVM中卡表很特殊，每512Byte大的内存页与卡表中一个Byte关联。
>
> 一般垃圾收集老年代中引用新生代对象的对象需要扫描整个老年代。这是需要卡表的原因。自写屏障清空后新生代所有对象都需要被创建(或者迁移)，因此不是脏内存页没有指向新生代的引用。这意味我们仅仅可以扫描脏内存页中的对象。