OopMap

前文我们说到，JVM 采用的可达性分析法有个缺点，就是从 GC Roots 找引用链耗时。

都说他耗时，他究竟耗时在哪里?

GC 进行扫描时，需要查看每个位置存储的是不是引用类型，如果是，其所引用的对象就不能被回收；如果不是，那就是基本类型，这些肯定是不会引用对象的；这种对 GC 无用的基本类型的数据非常多，每次 GC 都要去扫描，显然是非常浪费时间的。

而且迄今为止，所有收集器在 GC Roots 枚举这一步骤都是必须暂停用户线程的。

那有没有办法减少耗时呢?

一个很自然的想法，能不能用空间换时间? 把栈上的引用类型的位置全部记录下来，这样到 GC 的时候就可以直接读取，而不用一个个扫描了。Hotspot 就是这么实现的，这个用于存储引用类型的数据结构叫 OopMap。

OopMap 这个词可以拆成两部分：Oop 和 Map，Oop 的全称是 Ordinary Object Pointer 普通对象指针，Map 大家都知道是映射表，组合起来就是 普通对象指针映射表。

在 OopMap 的协助下，HotSpot 就能快速准确地完成 GC Roots 枚举啦。

安全点

OopMap 的更新，从直观上来说，需要在对象**引用关系发生变化**的时候修改。不过导致引用关系变化的指令非常多，如果对每条指令都记录 OopMap 的话 ，那将会需要大量的额外存储空间，空间成本就会变得无法忍受的高昂。选用一些特定的点来记录就能有效的缩小需要记录的数据量，这些特定的点就称为 **安全点 (Safepoint)。**

有了安全点，当 GC 回收需要停止用户线程的时候，将设置某个中断标志位，各个线程不断轮询这个标志位，发现需要挂起时，自己跑到最近的安全点，更新完 OopMap 才能挂

起。这主动式中断的方式是绝大部分现代虚拟机选择的方案，另一种抢占式就不介绍了。

安全点不是任意的选择，既不能太少以至于让收集器等待时间过长，也不能过多以至于过分增大运行时的内存负荷。通常选择一些执行时间较长的指令作为安全点，如**方法调用**、**循环跳转**和**异常跳转**等。

安全区域

使用安全点的设计似乎已经完美解决如何停顿用户线程，让虚拟机进入垃圾回收状态的问题了。但是，如果此时线程正处于 Sleep 或者 Blocked 状态，该怎么办？这些线程他不会自己走到安全点，就停不下来了。这个时候，安全点解决不了问题，需要引入 **安全区域 (Safe Region)**。

安全区域指的是，在某段代码中，**引用关系不会发生变化**，线程执行到这个区域是可以安全停下进行 GC 的。因此，我们也可以把 安全区域 看做是扩展的安全点。

当用户线程执行到安全区域里面的代码时，首先会标识自己已经进入了安全区域。那样当这段时间里虚拟机要发起 GC 时，就不必去管这些在安全区域内的线程了。当线程要离开安全区域时，它要检查虚拟机是否处于 STW 状态，如果是，则需要等待直到恢复。

总结

HotSpot 使用 OopMap 把引用类型的指针记录下来，让 GC Roots 的枚举变得快速准确。

为了减少更新 OopMap 的开销，引入了 **安全点**。GC STW 时，线程需要跑到距离自己最近的**安全点**，更新完 OopMap 才能挂起。

处于 Sleep 或者 Blocked 状态的线程无法跑到**安全点**，需要引入**安全区域**。GC 的时候，不会去管处于安全区域的线程，线程离开安全区域的时候，如果处于 STW 则需要等待直

至恢复。