链滴

Java 内存区域与内存溢出异常

作者：wgl530

原文链接：https://ld246.com/article/1578912532691

来源网站：链滴

Java内存区域与内存溢出异常

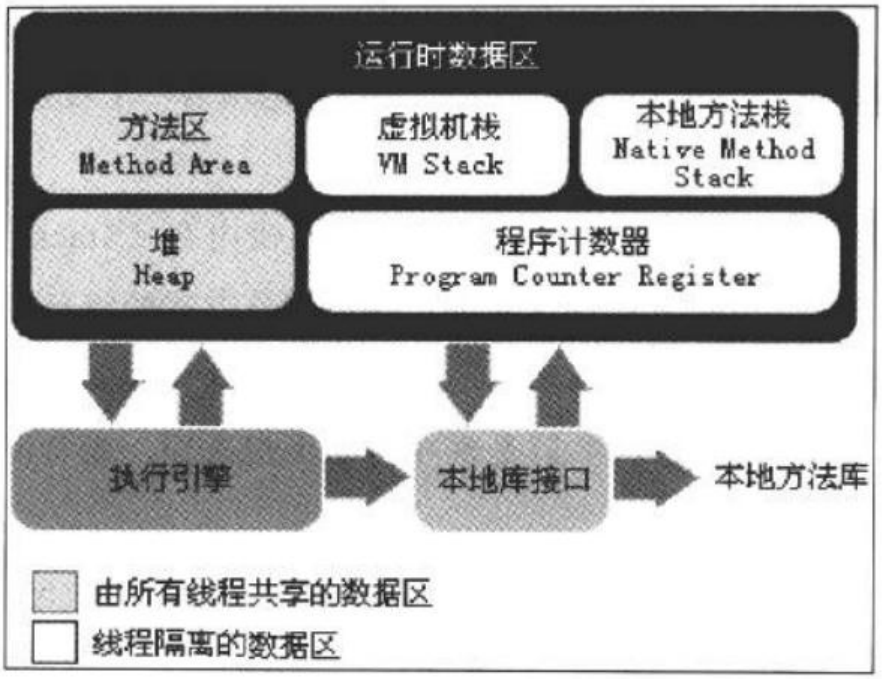
Java与C++之间有一堵由内存动态分配和垃圾收集技术所围成的“高墙”,墙外面的人想进去,墙里面人却想出来。

概述

对于从事C、C++程序开发的开发人员来说,在内存管理领域,他们既是拥有最高权力的“皇帝”又是从最基础工作的“劳动人民”—既拥有每一个对象的“所有权”,又担负着每一个对象生命开始到终结的护责任。对于Java程序员来说,在虚拟机自动内存管理机制的帮助下,不再需要为每一个new操作去写对的 delete/free代码,不容易出现内存泄漏和内存溢出问题,由虚拟机管理内存这一切看起来都很美好不过,也正是因为Java程序员把内存控制的权力交给了Java虚拟机,一旦出现内存泄漏和溢出方面的问题如果不了解虚拟机是怎样使用内存的,那么排查错误将会成为一项异常艰难的工作。

运行时数据区域

Java虚拟机在执行Java程序的过程中会把它所管理的内存划分为若干个不同的数据区域。这些区域都各自的用途,以及创建和销毁的时间,有的区域随着虚拟机进程的启动而存在,有些区域则依赖用户线程启动和结束而建立和销毁。根据《Java虚拟机规范(JavaSE7版)》的规定,Java虚拟机所管理的内存将包括以下几个运行时数据区域。



程序计数器

程序计数器(Program Counter Register)是一块较小的内存空间,它可以看作是当前线程所执行的字码的行号指示器。在虚拟机的概念模型里(仅是概念模型,各种虚拟机可能会通过一些更高效的方式去现),字节码解释器工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令,分支、循、跳转、异常处理、线程恢复等基础功能都需要依赖这个计数器来完成由于Java虚拟机的多线程是通线程轮流切换并分配处理器执行时间的方式来实现的在任何一个确定的时刻,一个处理器(对于多核处器来说是一个内核)都只会执行一条线程中的指令。

因此,为了线程切换后能恢复到正确的执行位置,每条线程都需要有一个独立的程序计数器,各条线程之计数器互不影响,独立存储,我们称这类内存区域为“线程私有”的内存。如果线程正在执行的是一个Ja

a方法,这个计数器记录的是正在执行的虚拟机字节码指令的地址;如果正在执行的是 Native方法,这个数器值则为空(Undefined)。此内存区域是唯一一个在Java虚拟机规范中没有规定任何 OutOfMemor Error情况的区域。

Java虚拟机栈

与程序计数器一样,Java虚拟机栈(Java Virtual Machine Stacks)也是线程私有的,它的生命周期与线相同。虚拟机栈描述的是Java方法执行的内存模型:每个方法在执行的同时都会创建一个栈帧(Stack Fr me)用于存储局部变量表、操作数栈、动态链接、方法出口等信息。每一个方法从调用直至执行完成过程,就对应着一个栈帧在虚拟机栈中入栈到出栈的过程。经常有人把Java内存区分为堆内存(Heap)栈内存(tack),这种分法比较粗糙,Java内存区域的划分实际上远比这复杂。

这种划分方式的流行只能说明大多数程序 员最关注的、与对象内存分配关系最密切的内存区域是这两。其中所指的“堆”笔者在后面会专门讲述,而所指的“栈”就是现在讲的虚拟机栈,或者说是虚拟机中局部变量表部分局部变量表存放了编译期可知的各种基本数据类型(boolean、byte、char、 shor 、int、foat、long、 double)、对象引用(reference类型,它不等同于对象本身,可能是一个指向对象始地址的引用指针,也可能是指向一个代表对象的句柄或其他与此对象相关的位置)和returnAddress类(指向了一条字节码指令的地址)。其中64位长度的long和 double类型的数据会占用2个局部变量空间(lot),其余的数据类型只占用1个。局部变量表所需的内存空间在编译期间完成分配,当进入一个方法时,个方法需要在帧中分配多大的局部变量空间是完全确定的,在方法运行期间不会改变局部变量表的大小Java虚拟机规范中,对这个区域规定了两种异常状况:如果线程请求的栈深度大于虚拟机所允许的深度,抛出 StackOverflow Error异常;如果虚拟机栈可以动态扩展(当前大部分的Java虚拟机都可动态扩展,不过Java虚拟机规范中也允许固定长度的虚拟机栈),如果扩展时无法申请到足够的内存,就会抛出 OutO Memory Error异常。