ZiWenXie

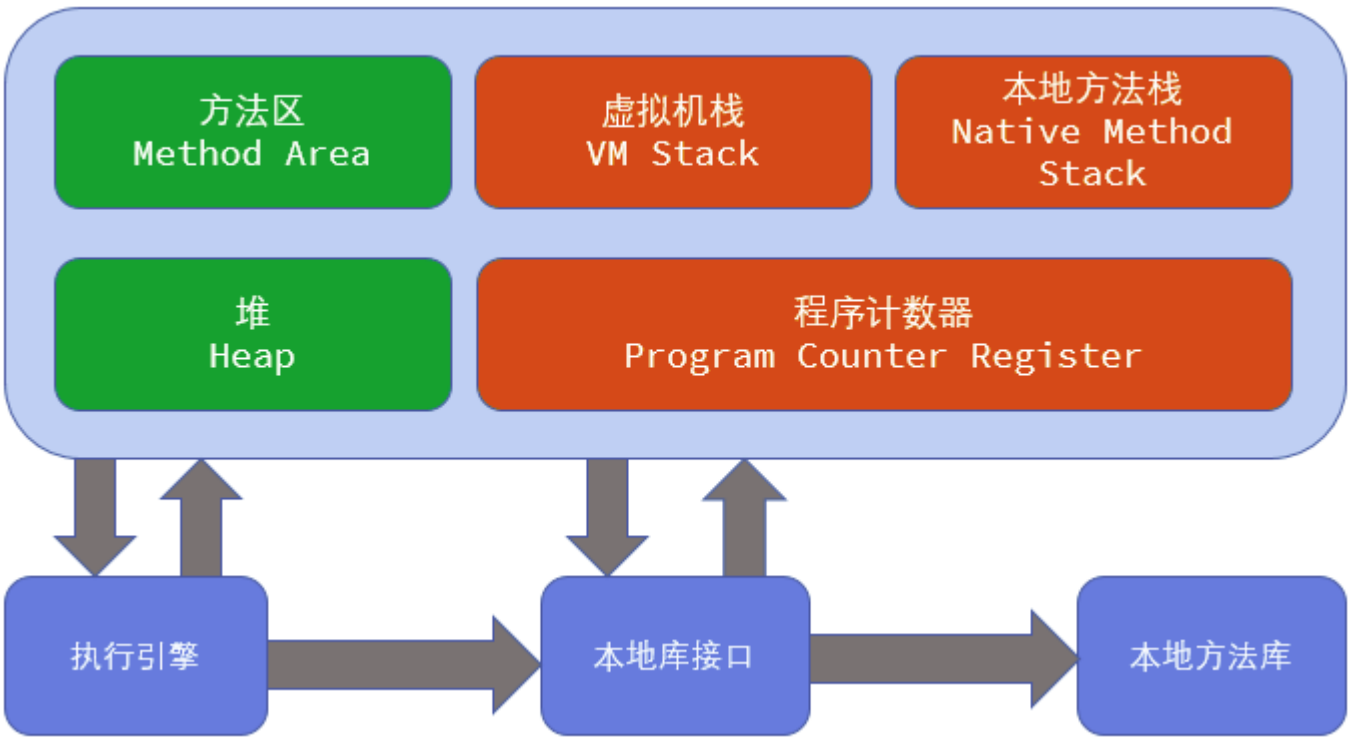
Leave something in internet

2017-06-01

JVM内存模型解析

引言

JVM内存模型可以分为两个部分，如下图所示，堆和方法区是所有线程共有的，而虚拟机栈，本地方法栈和程序计数器则是线程私有的。下面我们就来一一分析一下这些不同区域的作用。



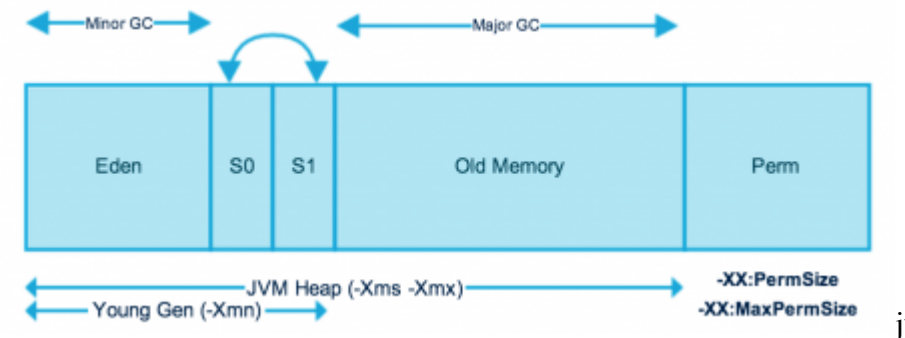
jvm-memory-model

JVM系列文章

1. JVM内存模型解析
2. JVM类加载机制详解

堆内存

堆内存是所有线程共有的，可以分为两个部分：年轻代和老年代。下图中的Perm代表的是永久代，但是注意永久代并不属于堆内存中的一部分，同时jdk1.8之后永久代也将被移除。



jvm-heap-memory

GC(垃圾回收器)对年轻代中的对象进行回收被称为Minor GC，用通俗一点的话说年轻代就是用来存放年轻的对象，年轻对象是什么意思呢？年轻对象可以简单的理解为没有经历过多次垃圾回收的对象，如果一个对象经历过了一定次数的Minor GC，JVM一般就会将这个对象放入到年老代，而JVM对年老代的对象的回收则称为Major GC。

如上图所示，年轻代中还可以细分为三个部分，我们需要重点关注这几点：

1. 大部分对象刚创建的时候，JVM会将其分布到Eden区域。
2. 当Eden区域中的对象达到一定的数目的时候，就会进行Minor GC，经历这次垃圾回收后所有存活的对象都会进入两个Suvivor Place中的一个。
3. 同一时刻两个Suvivor Place，即s0和s1中总有一个总是空的。
4. 年轻代中的对象经历过了多次的垃圾回收就会转移到年老代中。
5. 当申请不到空间时会抛出 OutOfMemoryError。下面我们简单的模拟一个堆内存溢出的情况：

```
import java.util.ArrayList;
import java.util.List;
/* java -Xms20m -Xmx20m HeapOOM */
public class HeapOOM {
    static class OOMObject {
    }
    public static void main(String[] args) {
        List<OOMObject> list = new ArrayList<OOMObject>();
        while (true) {
            list.add(new OOMObject());
        }
    }
}
```

下面是执行结果：

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOf(Arrays.java:3210)
        at java.util.Arrays.copyOf(Arrays.java:3181)
        at java.util.ArrayList.grow(ArrayList.java:261)
        at java.util.ArrayList.ensureExplicitCapacity(ArrayList.java:235)
        at java.util.ArrayList.ensureCapacityInternal(ArrayList.java:227)
        at java.util.ArrayList.add(ArrayList.java:458)
        at HeapOOM.main(HeapOOM.java:13)
```

堆内存是我们平时在生产环境中进行性能调优中的一个非常重要的部分，对于这里我在另外一篇文章JVM垃圾回收算法及回收器详解有详细介绍，这里我们还是拓展补充几个常见的性能调优参数：

- PretenureSizeThreshold: 直接晋升到老年代的对象大小，设置这个参数后，大于这个参数的对象将直接在老年代分配。
- MaxTenuringThrehold: 晋升到老年代的对象年龄。每个对象在坚持过一次Minor GC之后，年龄就会加1，当超过这个参数值时就进入老年代。
- UseAdaptiveSizePolicy: 动态调整Java堆中各个区域的大小以及进入老年代的年龄。
- SurvivorRattio: 新生代Eden区域与Survivor区域的容量比值，默认为8，代表Eden: Suvivor= 8: 1。
- NewRatio: 设置新生代（包括Eden和两个Survivor区）与老年代的比值（除去持久代），设置为3，则新生代与老年代所占比值为1：3，新生代占整个堆栈的1/4。
- Xmx: 设置JVM堆最大内存。
- Xms: 设置JVM堆初始内存。
- Xmn: 参数设置了年轻代内存。

方法区

方法区与Java堆一样，是各个线程共享的区域，它用于存储已被虚拟机加载的类信息，常量，静态变量，即时编译(JIT)后的代码等数据。

对于JDK1.8之前的HotSpot虚拟机而言，很多人经常将方法区称为我们上图中所描述的永久代，实际上两者并不等价，因为这仅仅是HotSpot的设计团队选择利用永久代来实现方法区而言。同时对于其他虚拟机比如IBM J9中是不存在永久代的概念的。

其实，移除永久代的工作从JDK1.7就开始了。JDK1.7中，存储在永久代的部分数据就已经转移到了Java Heap或者是 Native Heap。但永久代仍存在于JDK1.7中，并没完全移除，譬如符号引用(Symbols)转移到了native heap；字面量(interned strings)转移到了java heap；类的静态变量(class statics)转移到了java heap，而在JDK1.8之后永久代的该概念也已经不再存在取而代之的是元空间metaspace。

常量池其实是方法区中的一部分，因为这里比较重要，所以我们拿出来单独看一下。注意我们这里所说的运行时的常量池并仅仅是指Class文件中的常量池，因为JVM可能会进行即时编译进行优化，在运行时将部分常量载入到常量池中。

程序计数器

JVM中的程序计数器和计算机组成原理中提到的程序计数器PC概念类似，是线程私有的，用来记录当前执行的字节码位置。还是稍微解释一下吧，CPU的占有时间是以分片的形式分配给给每个不同线程的，从操作系统的角度来讲，在不同线程之间切换的时候就是依赖程序计数器来记录上一次线程所执行到具体的代码的行数，在JVM就是字节码。

Java虚拟机栈

与程序计数器一样，Java虚拟机栈也是线程私有的，用通俗的话将它就是我们常常听说到堆栈中的那个“栈内存”。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 Frame)用于存储局部变量表（局部变量表需要的内存在编译期间就确定了所以在方法运行期间不会改变大小），操作数栈，动态链接，方法出口等信息。每一个方法从调用至出栈的过程，就对应着栈帧在虚拟机中从入栈到出栈的过程。p.s: 关于栈帧这里我们以后讲虚拟机字节码执行引擎的时候再来仔细分析。