

第25讲 | 谈谈JVM内存区域的划分，哪些区域可能发生OutOfMemoryError?

2018-07-03 杨晓峰

Java核心技术36讲

[进入课程 >](#)



讲述：黄洲君

时长 10:42 大小 4.90M



今天，我将从内存管理的角度，进一步探索 Java 虚拟机（JVM）。垃圾收集机制为我们打理了很多繁琐的工作，大大提高了开发的效率，但是，垃圾收集也不是万能的，懂得 JVM 内部的内存结构、工作机制，是设计高扩展性应用和诊断运行时问题的基础，也是 Java 工程师进阶的必备能力。

今天我要问你的问题是，谈谈 JVM 内存区域的划分，哪些区域可能发生 OutOfMemoryError？

典型回答

通常可以把 JVM 内存区域分为下面几个方面，其中，有的区域是以线程为单位，而有的区域则是整个 JVM 进程唯一的。

首先，**程序计数器**（PC，Program Counter Register）。在 JVM 规范中，每个线程都有它自己的程序计数器，并且任何时间一个线程都只有一个方法在执行，也就是所谓的当前方法。程序计数器会存储当前线程正在执行的 Java 方法的 JVM 指令地址；或者，如果是在执行本地方法，则是未指定值（undefined）。

第二，**Java 虚拟机栈**（Java Virtual Machine Stack），早期也叫 Java 栈。每个线程在创建时都会创建一个虚拟机栈，其内部保存一个个的栈帧（Stack Frame），对应着一次次的 Java 方法调用。

前面谈程序计数器时，提到了当前方法；同理，在一个时间点，对应的只会有一个活动的栈帧，通常叫作当前帧，方法所在的类叫作当前类。如果在该方法中调用了其他方法，对应的新的栈帧会被创建出来，成为新的当前帧，一直到它返回结果或者执行结束。JVM 直接对 Java 栈的操作只有两个，就是对栈帧的压栈和出栈。

栈帧中存储着局部变量表、操作数（operand）栈、动态链接、方法正常退出或者异常退出的定义等。

第三，**堆**（Heap），它是 Java 内存管理的核心区域，用来放置 Java 对象实例，几乎所有创建的 Java 对象实例都是被直接分配在堆上。堆被所有的线程共享，在虚拟机启动时，我们指定的“Xmx”之类参数就是用来指定最大堆空间等指标。

理所当然，堆也是垃圾收集器重点照顾的区域，所以堆内空间还会被不同的垃圾收集器进行进一步的细分，最有名的就是新生代、老年代的划分。

第四，**方法区**（Method Area）。这也是所有线程共享的一块内存区域，用于存储所谓的元（Meta）数据，例如类结构信息，以及对应的运行时常量池、字段、方法代码等。

由于早期的 Hotspot JVM 实现，很多人习惯于将方法区称为永久代（Permanent Generation）。Oracle JDK 8 中将永久代移除，同时增加了元数据区（Metaspace）。

第五，**运行时常量池**（Run-Time Constant Pool），这是方法区的一部分。如果仔细分析过反编译的类文件结构，你能看到版本号、字段、方法、超类、接口等各种信息，还有一项

信息就是常量池。Java 的常量池可以存放各种常量信息，不管是编译期生成的各种字面量，还是需要在运行时决定的符号引用，所以它比一般语言的符号表存储的信息更加宽泛。

第六，**本地方法栈**（Native Method Stack）。它和 Java 虚拟机栈是非常相似的，支持对本地方法的调用，也是每个线程都会创建一个。在 Oracle Hotspot JVM 中，本地方法栈和 Java 虚拟机栈是在同一块儿区域，这完全取决于技术实现的决定，并未在规范中强制。

考点分析

这是个 JVM 领域的基础题目，我给出的答案依据的是 [JVM 规范](#) 中运行时数据区定义，这也和大多数书籍和资料解读的角度类似。

JVM 内部的概念庞杂，对于初学者比较晦涩，我的建议是在工作之余，还是要去阅读经典书籍，比如我推荐过多次的《深入理解 Java 虚拟机》。

今天这一讲作为 Java 虚拟机内存管理的开篇，我会侧重于：

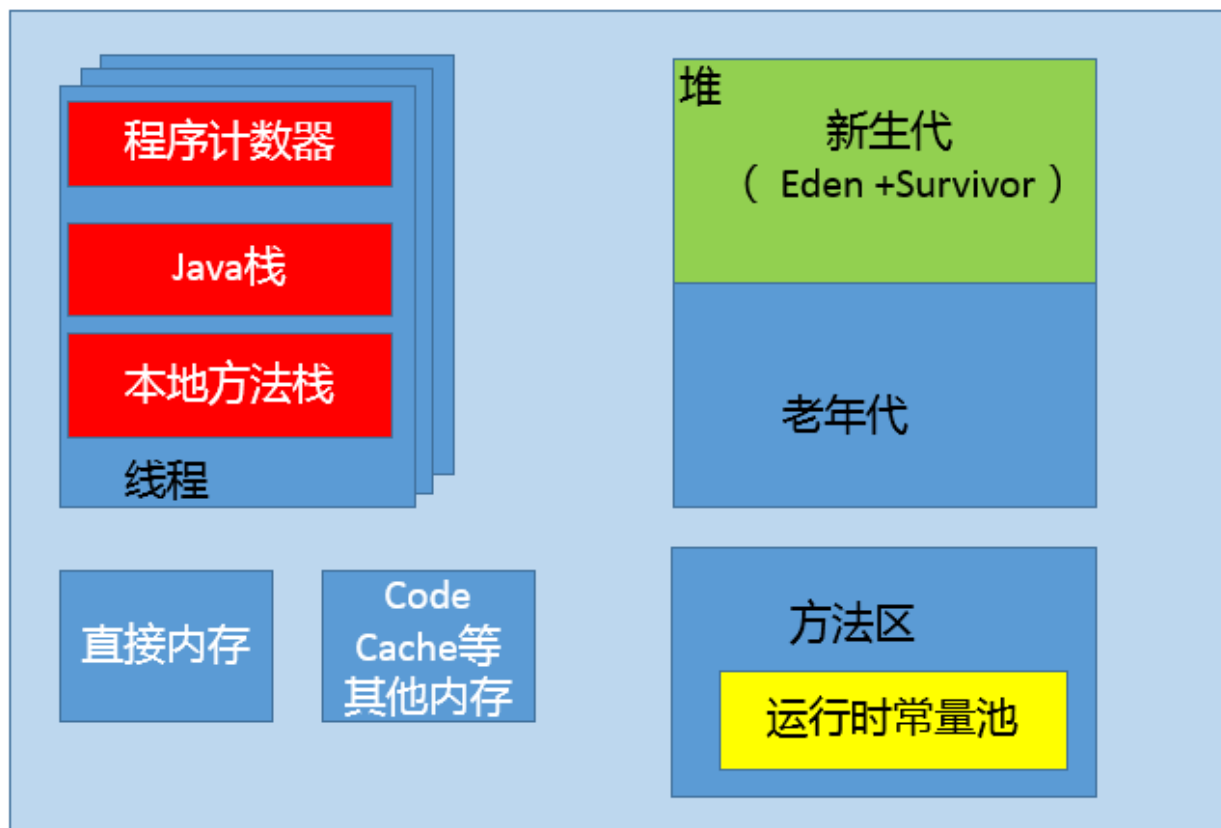
分析广义上的 JVM 内存结构或者说 Java 进程内存结构。

谈到 Java 内存模型，不可避免的要涉及 OutOfMemory（OOM）问题，那么在 Java 里面存在哪些种 OOM 的可能性，分别对应哪个内存区域的异常状况呢？

注意，具体 JVM 的内存结构，其实取决于其实现，不同厂商的 JVM，或者同一厂商发布的不同版本，都有可能存在一定差异。我在下面的分析中，还会介绍 Oracle Hotspot JVM 的部分设计变化。

知识扩展

首先，为了让你有个更加直观、清晰的印象，我画了一个简单的内存结构图，里面展示了我前面提到的堆、线程栈等区域，并从数量上说明了什么是线程私有，例如，程序计数器、Java 栈等，以及什么是 Java 进程唯一。另外，还额外划分出了直接内存等区域。



这张图反映了实际中 Java 进程内存占用，与规范中定义的 JVM 运行时数据区之间的差别，它可以看作是运行时数据区的一个超集。毕竟理论上的视角和现实中的视角是有区别的，规范侧重的是通用的、无差别的部分，而对于应用开发者来说，只要是 Java 进程在运行时占用，都会影响到我们的工程实践。

我这里简要介绍两点区别：

直接内存（Direct Memory）区域，它就是在[专栏第 12 讲](#)中谈到的 Direct Buffer 所直接分配的内存，也是个容易出现问题的地方。尽管，在 JVM 工程师的眼中，并不认为它是 JVM 内部内存的一部分，也并未体现 JVM 内存模型中。

JVM 本身是个本地程序，还需要其他的内存去完成各种基本任务，比如，JIT Compiler 在运行时对热点方法进行编译，就会将编译后的方法储存在 Code Cache 里面；GC 等功能需要运行在本地线程之中，类似部分都需要占用内存空间。这些是实现 JVM JIT 等功能的需要，但规范中并不涉及。

如果深入到 JVM 的实现细节，你会发现一些结论似乎有些模棱两可，比如：

Java 对象是不是都创建在堆上的呢？

我注意到有一些观点，认为通过[逃逸分析](#)，JVM 会在栈上分配那些不会逃逸的对象，这在理论上是可行的，但是取决于 JVM 设计者的选择。据我所知，Oracle Hotspot JVM 中并未这么做，这一点在逃逸分析相关的[文档](#)里已经说明，所以可以明确所有的对象实例都是创建在堆上。

目前很多书籍还是基于 JDK 7 以前的版本，JDK 已经发生了很大变化，Intern 字符串的缓存和静态变量曾经都被分配在永久代上，而永久代已经被元数据区取代。但是，Intern 字符串缓存和静态变量并不是被转移到元数据区，而是直接在堆上分配，所以这一点同样符合前面一点的结论：对象实例都是分配在堆上。


接下来，我们来看看什么是 OOM 问题，它可能在哪些内存区域发生？

首先，OOM 如果通俗点儿说，就是 JVM 内存不够用了，javadoc 中对[OutOfMemoryError](#)的解释是，没有空闲内存，并且垃圾收集器也无法提供更多内存。

这里面隐含着一层意思是，在抛出 OutOfMemoryError 之前，通常垃圾收集器会被触发，尽其所能去清理出空间，例如：

我在[专栏第 4 讲](#)的引用机制分析中，已经提到了 JVM 会去尝试回收软引用指向的对象等。

在[java.nio.Bits.reserveMemory\(\)](#) 方法中，我们能清楚的看到，System.gc() 会被调用，以清理空间，这也是为什么在大量使用 NIO 的 Direct Buffer 之类时，通常建议不要加下面的参数，毕竟是个最后的尝试，有可能避免一定的内存不足问题。

 复制代码

```
1 -XX:+DisableExplicitGC
```

当然，也不是在任何情况下垃圾收集器都会被触发的，比如，我们去分配一个超大对象，类似一个超大数组超过堆的最大值，JVM 可以判断出垃圾收集并不能解决这个问题，所以直接抛出 OutOfMemoryError。

从我前面分析的数据区的角度，除了程序计数器，其他区域都有可能会因为可能的空间不足发生 OutOfMemoryError，简单总结如下：

堆内存不足是最常见的 OOM 原因之一，抛出的错误信息是 “java.lang.OutOfMemoryError:Java heap space”，原因可能千奇百怪，例如，可能存在内存泄漏问题；也很有可能就是堆的大小不合理，比如我们要处理比较可观的数据量，但是没有显式指定 JVM 堆大小或者指定数值偏小；或者出现 JVM 处理引用不及时，导致堆积起来，内存无法释放等。

而对于 Java 虚拟机栈和本地方法栈，这里要稍微复杂一点。如果我们写一段程序不断的进行递归调用，而且没有退出条件，就会导致不断地进行压栈。类似这种情况，JVM 实际会抛出 StackOverflowError；当然，如果 JVM 试图去扩展栈空间的时候失败，则会抛出 OutOfMemoryError。

对于老版本的 Oracle JDK，因为永久代的大小是有限的，并且 JVM 对永久代垃圾回收（如，常量池回收、卸载不再需要的类型）非常不积极，所以当我们不断添加新类型的时候，永久代出现 OutOfMemoryError 也非常多见，尤其是在运行时存在大量动态类型生成的场合；类似 Intern 字符串缓存占用太多空间，也会导致 OOM 问题。对应的异常信息，会标记出来和永久代相关：“java.lang.OutOfMemoryError: PermGen space”。

随着元数据区的引入，方法区内存已经不再那么窘迫，所以相应的 OOM 有所改观，出现 OOM，异常信息则变成了：“java.lang.OutOfMemoryError: Metaspace”。

直接内存不足，也会导致 OOM，这个已经[专栏第 11 讲](#)介绍过。

今天是 JVM 内存部分的第一讲，算是我们先进行了热身准备，我介绍了主要的内存区域，以及在不同版本 Hotspot JVM 内部的变化，并且分析了各区域是否可能产生 OutOfMemoryError，以及 OOME 发生的典型情况。

一课一练

关于今天我们讨论的题目你做到心中有数了吗？今天的思考题是，我在试图分配一个 100M bytes 大数组的时候发生了 OOME，但是 GC 日志显示，明明堆上还有远不止 100M 的空间，你觉得可能问题的原因是什么？想要弄清楚这个问题，还需要什么信息呢？

请你在留言区写写你对这个问题的思考，我会选出经过认真思考的留言，送给你一份学习奖励礼券，欢迎你与我一起讨论。

你的朋友是不是也在准备面试呢？你可以“请朋友读”，把今天的题目分享给好友，或许你能帮到他。

Java 核心技术36讲

—— 前 Oracle 首席工程师
带你修炼 Java 内功 ——

杨晓峰 前 Oracle 首席工程师



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 第24讲 | 有哪些方法可以在运行时动态生成一个Java类？

下一篇 第26讲 | 如何监控和诊断JVM堆内和堆外内存使用？

精选留言 (39)

写留言



I am a ...

2018-07-03

104

如果仅从jvm的角度来看，要看下新生代和老年代的垃圾回收机制是什么。如果新生代是serial，会默认使用copying算法，利用两块eden和survivor来进行处理。但是默认当遇到超大对象时，会直接将超大对象放置到老年代中，而不用走正常对象的存活次数记录。因为要放置的是一个byte数组，那么必然需要申请连续的空间，当空间不足时，会进行gc操作。这里又需要看老年代的gc机制是哪一种。如果是serial old，那么会采用mark...

展开 ▾

作者回复: 非常不错的总结





石头狮子

2018-07-03

👍 15

1, 新生代大小过小。无法分配足够的内存。同时也老年代过小, 导致提升失败。这时系统认为没有足够的空间存放该100M数据。

2, 栈可以抽象的看成计算资源。堆看成存储资源。计算资源不共享, 不会发生线程安全问题。堆资源共享, 容易发生线程安全问题。...

展开 ▾



鸡肉饭饭

2018-07-03

👍 10

我们拿JDK7来说, 有可能的原因是JVM的剩余内存有100M, 但是它是分在不同年龄代的内存区域。

因此应当单独的去查看每一块eden, survivor, old的大小, (通过SurvivorRatio知道s和e的比例大小, 通过MaxNewSize知道young和old的比例)看看这三块区域是否有超过100...

展开 ▾



LenX

2018-07-03

👍 8

从不同的垃圾收集器角度来看:

首先, 数组的分配是需要连续的内存空间的(据说, 有个别非主流JVM支持大数组用不连续的内存空间分配☹)。所以:

...

展开 ▾

作者回复: 很好的视角, g1 region之类确实有影响, 另外g1还是有年代的概念的



爱吃芒果的...

2018-07-03

👍 6

因为给数组分配的是连续地址, 而显示的是总的地址, 不管是不是连续的。

作者回复: 也对, 最好综合考虑堆内存结构、gc区别等, 后续会讲解





tyson

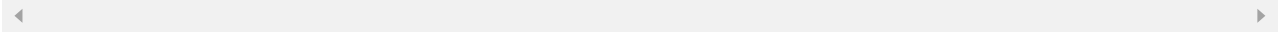
2018-07-03

👍 5

堆内存100M 包含了新生代(eden+s0+1)和老年代，大对象一般分配在老年代，那么最有可能在分配过程中老年代的空间不足。

展开 ∨

作者回复: 不错，可能性很多，其实和gc的选择也有关，例如g1 region比较小



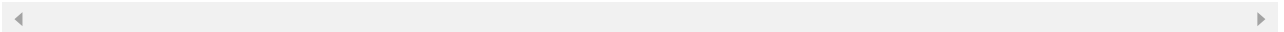
任鹏斌

2018-08-06

👍 4

老师既然元数据区也存在溢出，那么为什么要用元数据区替换永久代呢，有什么好处吗？

作者回复: metaspace 默认是自增的，永久带做不到



markin

2018-07-08

👍 3

老师，能否跟我们介绍一下您平时获取资料的渠道。比如apache的一些开源项目，官网上就有很丰富的文档。但是我们获取jvm相关文档的渠道少之又少，无非就是博客或者书籍，这些都比较繁杂，并且可能参杂着很多难以识别的错误观点。授人以鱼不如授人以渔，先谢谢老师了。

展开 ∨

作者回复: Oracle官网也提供了很多好的文档：

虚拟机规范 <https://docs.oracle.com/javase/specs/jvms/se8/html/index.html>

诊断指南

<https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/index.html>

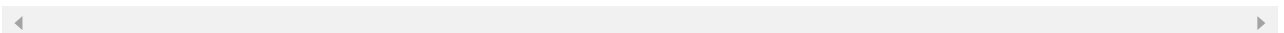
调优指南 <https://docs.oracle.com/javase/10/gctuning/>

Openjdk网站，或者那些感兴趣的邮件列表 <http://mail.openjdk.java.net/mailman/listinfo>

YouTube上查查javaone，JVM summit之类

回头有必要整理个书单之类

但这些东西太多了，自己把握一下





鹅米豆发

2018-07-03

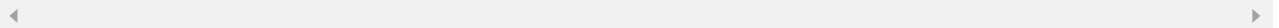
👍 3

可能一，新生代没有足够的连续空间，且不能直接在老年代分配。比如 $E+S_0+S_1 > 100MB$ ，但 $E < 100MB$ ， $S_0 < 100MB$ 。

可能二，大对象直接进入老年代，但老年代也没有足够的连续空间。参数 `+XX:PretenureSizeThreshold`。...

展开 ▾

作者回复: 不错，下一章会有更多内存结构细节



夏洛克的救...

2019-01-14

👍 2

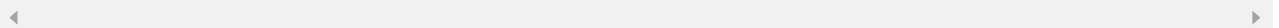
Tomcat运行中突然出现`java.lang.OutOfMemoryError: PermGen space`有什么工具可以排查原因吗？

作者回复: 简单点处理，可以：

先看看永久带给了多大，如果太小，可以适当增大，使用`'-XX:MaxPermSize=NNNm'`;

如果没开启classunloading，可以根据GC选项做配置，例如，如果使用的CMS，可以加上 `"-XX:+CMSClassUnloadingEnabled -XX:+CMSPermGenSweepingEnabled"`

通常就能解决问题了，如果还是有问题，那就要看看是不是出现了classloader leak，常见做法如，取Heap dump，然后用类似Eclipse MAT这样的工具，看看有没有不回收的自定义classloader实例之类



师琳博

2018-07-03

👍 2

100m的byte数组，一个byte对应一个引用，这样需要100m个的引用，所以需要的栈空间也不会低于100m,而对象的引用是在栈中分配的，(栈和堆加起来估计不低于200m)况且还是数组，对应的那么多引用还需要分配连续的内存空间，堆空间够的话，个人认为可能是栈空间不足造成的

展开 ▾



刘p辉

2018-07-30

👍 1

1.首先堆内存是分代的，总的内存超过100M，不能保证新生代，老年代都有足够的内存。
2.为对象，数组分配内存需要连续的内存空间，有可能堆的总内存远超过要分配内存大小，但是在即使进行过垃圾回收（标记整理）后还是不存在足够的连续内存空间就会OOM。

展开 ▾



Geek_13514...

2018-07-29

👍 1

请问compressed class space区域怎么理解？是metaspace的一部分吗？如果是的话，有些采集工具为何会把它占的大小单独显示出来呢？而不直接显示metaspace的大小呢



Steven°°...

2018-07-04

👍 1

数组是连续分配的，gc表明有多余100m，但有可能满足不了连续100m的空间，故会报OOM



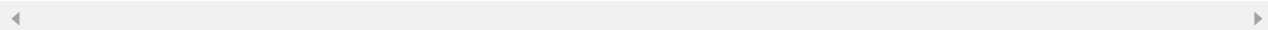
boom

2018-07-03

👍 1

小白请教一个不相关的问题~java 内存模型跟 jvm内存模型的区别与联系是啥呢~

作者回复: 怎么叫都有，看上下文，jsr133 jmm是解决多线程环境一致性，或者可以看做memory ordering model



三口先生

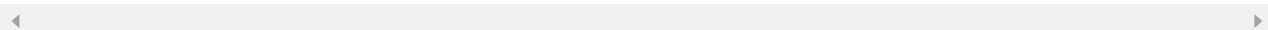
2018-07-03

👍 1

堆内存比例设置不合理

展开 ▾

作者回复: 也对，回答比较简洁，哈哈





sunlight00...

2018-07-03

👍 1

堆上有空间的划分，新生代和老年代，有可能新生代的空间不够，看到的是老年代的空间，个人猜测😊



未完的歌

2018-07-03

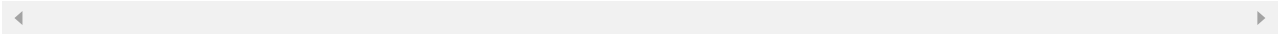
👍 1

有可能是内存碎片化问题，或者是大对象的内存分配策略问题。

需要了解一下积极垃圾收集算法，例如Mark Sweep就会造成内存碎片化问题，另外内存分配策略也是一个关注点。

展开 ∨

作者回复: 是的



鲸息

2019-03-14

👍

老年代碎片化了。需要查看老年代的内存使用状况，对于 CMS 可以间接地看是否打开了每次 CMS GC 以后就立刻做一次碎片整理的开关。

展开 ∨



刘胜

2019-03-12

👍

不知道，无法解答。可能是超过了jvm默认可以处置的内存的大小了吧？我只能猜了。