JVM - 走进方法区的前世今生

作者：douniwan
原文链接：https://ld246.com/article/1576677568923
来源网站：链滴

什么是方法区?

方法区（Method area）是可供各个线程共享的运行时内存区域，它存储了每一个类的结构信息，例：运行时常量池（Runtime constant pool）。字段和方法数据、构造函数和普通方法的字节码内容还包括一些在类、实例、接口初始化时用到的特殊方法。

方法区在虚拟机启动的时候创建，方法区的容量可以是固定的，也可以随着程序的执行实现动态扩展并在不需要过多空间的时候自动回收。方法区在实际内存中可以是不连续的。

方法区可以发生如下异常：

如果方法区的内存空间不能满足内存分配请求，那么Java虚拟机将抛出一个OutOfMemoryError。

以上就是Java虚拟机规范中对方法区的一个定义，它描述的是一种规范，而不是一种具体的实现。因在不同的虚拟机中方法区存在与不同的位置。我们这里将主要阐述Hotspot虚拟机中的方法区实现。

这里我们将方法区的历史分割成三个部分，以此来分别讲述这之间发生了什么。这三个部分分别是**JDK7之前**、**JDK7**和**JDK8及以后**。

JDK7之前

在JDK之前，方法区位于永久代（PermGen），永久代和堆相互隔离，永久代的大小在JVM中可以设一个固定值，不可变。

永久代是Hotspot虚拟机特有的概念，是方法区的一种实现，别的JVM都没有这个东西。永久代与新代和老年代相样，前者并不是位于堆中，后后两者是位于堆中的。

在这个时候永久代就是方法区的实现，这个时期默认方法区即永久代。在这个时候永久代里面存放了多东西，例如，符号引用（Symbols）、字符串常量池（interned strings）、类的静态变量（class static variables）、运行时常量池以及其它信息。由于这个时候方法区是由永久代实现的，那么方法区现异常后会抛出这样的信息：java.lang.OutOfMemoryError: PermGen， 这里的 PermGen 就是永久的意思，从这个就可以看出此时的方法区的实现是永久代。

JDK7

在这个时候官方以及发现用永久代实现方法区容易导致内存泄漏的问题了，同时为了后面将Hotspot拟机与其他虚拟机整合，已经有将方法区改用其他的方式类实现了，但是并没有动工！此时只是将原位于**永久代中的字符串常量、类的静态变量池转移到了Java Heap中，还有符号引用转移到了Native Memory**

此时你使用String类的intern()方法，你会发现与jdk6及以前出现不一样的 结果。

在 Jdk6 以及以前的版本中，字符串的常量池是放在堆的Perm区的，Perm区是一个类静态的区域，要存储一些加载类的信息，常量池，方法片段等内容，默认大小只有4m，一旦常量池中大量使用 intern 是会直接产生java.lang.OutOfMemoryError:PermGen space错误的。

JDK8及以后

取消永久代，方法区由元空间（Metaspace）实现，元空间仍然与堆不相连，但与堆共享物理内存，辑上可认为在堆中。

元空间的本质和永久代类似，都是对JVM规范中方法区的实现。不过**元空间与永久代之间最大的区别于：元空间并不在虚拟机中，而是使用本地内存。**，理论上取决于32位/64位系统可虚拟的内存大小可见也不是无限制的，需要配置参数。

在之前的永久代实现中，如果要修改方法区的大小配置，需要使用~~PermSize，MaxPermSize~~数，而改为元空间之后，就需要使用MetaspaceSize，MaxMetaspaceSize。

为什么移除永久代?

- 字符串存在永久代中，容易出现性能问题和内存溢出。
- 永久代大小不容易确定，PermSize指定太小容易造成永久代OOM
- 永久代会为 GC 带来不必要的复杂度，并且回收效率偏低。
- Oracle 可能会将HotSpot 与 JRockit 合二为一。

在JDK1.7中, 已经把原本放在永久代的字符串常量池移出, 放在堆中. 为什么这样做呢?

因为使用永久代来实现方法区不是个好主意, 很容易遇到内存溢出的问题. 我们通常使用PermSize和MxPermSize设置永久代的大小, 这个大小就决定了永久代的上限, 但是我们不是总是知道应该设置为多的, 如果使用默认值容易遇到OOM错误。

类的元数据, 字符串池, 类的静态变量将会从永久代移除, 放入Java heap或者native memory。其中议JVM的实现中将类的元数据放入 native memory, 将字符串池和类的静态变量放入java堆中. 这样以加载多少类的元数据就不在由MaxPermSize控制, 而由系统的实际可用空间来控制.

为什么这么做呢? 减少OOM只是表因, 更深层的原因还是要合并HotSpot和JRockit的代码, JRockit从没有一个叫永久代的东西, 但是运行良好, 也不需要开发运维人员设置这么一个永久代的大小。当然不担心运行性能问题了, 在覆盖到的测试中, 程序启动和运行速度降低不超过1%, 但是这一点性能损失换了更大的安全保障。

本文参考：

《深入理解Java虚拟机》

《Java虚拟机规范（Java SE 8 版）》

   JDK1.8之前和之后的方法区