

加微信：642945106 发送“赠送”领取赠送精品课程



发数字“2”获取众筹列表

下载APP



## 03 | Java虚拟机原理：JVM为什么被称为机器（machine）？

2019-11-22 李智慧

后端技术基础详解

进入课程 >



讲述：李智慧

时长 15:31 大小 14.21M



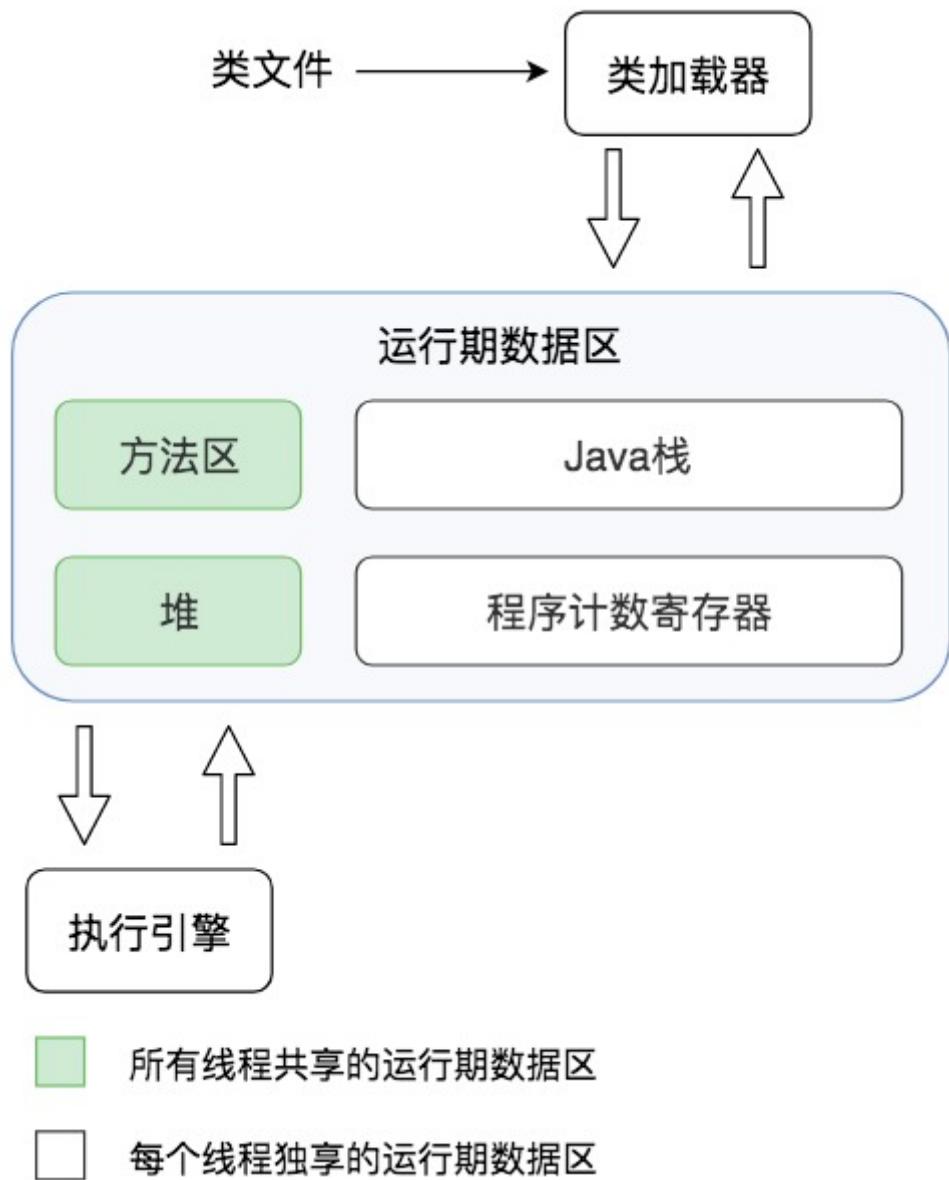
人们常说，Java 是一种跨平台的语言，这意味着 Java 开发出来的程序经过编译后，可以在 Linux 上运行，也可以在 Windows 上运行；可以在 PC、服务器上运行，也可以在手机上运行；可以在 X86 的 CPU 上运行，也可以在 ARM 的 CPU 上运行。

因为不同操作系统，特别是不同 CPU 架构，是不可能执行相同的指令的。而 Java 之所以有这种神奇的特性，就是因为 Java 编译的字节码文件不是直接在底层的系统平台上运行的，而是在 Java 虚拟机 JVM 上运行，JVM 屏蔽了底层系统的不同，为 Java 字节码文件构造了一个统一的运行环境。JVM 本质上也是一个应用程序，启动以后加载执行 Java 字节码文件。JVM 的全称是 Java Virtual Machine，你有没有想过，这样一个程序为什么被称为机器（Machine）呢？

其实，如果回答了这个问题，也就了解了 JVM 的底层构造了。这样在进行 Java 开发的时候，如果遇到各种问题，都可以思考一下在 JVM 层面是如何的？然后进一步查找资料、分析问题，直至真正地解决问题。

## JVM 的组成构造

要想知道这个问题的答案，我们首先需要了解 JVM 的构造。JVM 主要由类加载器、运行时数据区、执行引擎三个部分组成。



运行时数据区主要包括方法区、堆、Java 栈、程序计数寄存器。

方法区主要存放从磁盘加载进来的类字节码，而在程序运行过程中创建的类实例则存放在堆里。程序运行的时候，实际上是以线程为单位运行的，当 JVM 进入启动类的 main 方法的时候，就会为应用程序创建一个主线程，main 方法里的代码就会被这个主线程执行，每个

线程有自己的 Java 栈，栈里存放着方法运行期的局部变量。而当前线程执行到哪一行字节码指令，这个信息则被存放在程序计数寄存器。

一个典型的 Java 程序运行过程是下面这样的。

通过 Java 命令启动 JVM，JVM 的类加载器根据 Java 命令的参数到指定的路径加载.class 类文件，类文件被加载到内存后，存放在专门的方法区。然后 JVM 创建一个主线程执行这个类文件的 main 方法，main 方法的输入参数和方法内定义的变量被压入 Java 栈。如果在方法内创建了一个对象实例，这个对象实例信息将被存放到堆里，而对象实例的引用，也就是对象实例在堆中的地址信息则会被记录在栈里。堆中记录的对象实例信息主要是成员变量信息，因为类方法内的可执行代码存放在方法区，而方法内的局部变量存放在线程的栈里。

程序计数寄存器一开始存放的是 main 方法的第一行代码位置，JVM 的执行引擎根据这个位置去方法区的对应位置加载这行代码指令，将其解释为自身所在平台的 CPU 指令后交给 CPU 执行。如果在 main 方法里调用了其他方法，那么在进入其他方法的时候，会在 Java 栈中为这个方法创建一个新的栈帧，当线程在这个方法内执行的时候，方法内的局部变量都存放在这个栈帧里。当这个方法执行完毕退出的时候，把这个栈帧从 Java 栈中出栈，这样当前栈帧，也就是堆栈的栈顶就又回到了 main 方法的栈帧，使用这个栈帧里的变量，继续执行 main 方法。这样，即使 main 方法和 f 方法都定义相同的变量，JVM 也不会弄错。这部分内容我们在第一篇已经讨论过，JVM 作为一个 machine，和操作系统的处理线程栈的方法是一样的。



Java 的线程安全常常让人困惑，你可以试着从 Java 栈的角度去理解，所有在方法内定义的基本类型变量，都会被每个运行这个方法的线程放入自己的栈中，线程的栈彼此隔离，所以这些变量一定是线程安全的。如果在方法里创建了一个对象实例，这个对象实例如果没有被

方法返回或者放入某些外部的对象容器中的话，也就是说这个对象的引用没有离开这个方法，虽然这个对象被放置在堆中，但是这个对象不会被其他线程访问到，也是线程安全的。

相反，像 Servlet 这样的类，在 Web 容器中创建以后，会被传递给每个访问 Web 应用的用户线程执行，这个类就不是线程安全的。但这并不意味着一定会引发线程安全问题，如果 Servlet 类里没有成员变量，即使多线程同时执行这个 Servlet 实例的方法，也不会造成成员变量冲突。这种对象被称作无状态对象，也就是说对象不记录状态，执行这个对象的任何方法都不会改变对象的状态，也就不会有线程安全问题了。事实上，Web 开发实践中，常见的 Service 类、DAO 类，都被设计成无状态对象，所以虽然我们开发的 Web 应用都是多线程的应用，因为 Web 容器一定会创建多线程来执行我们的代码，但是我们开发中却可以很少考虑线程安全的问题。

我们再回过头看 JVM，它封装了一组自定义的字节码指令集，有自己的程序计数器和执行引擎，像 CPU 一样，可以执行运算指令。它还像操作系统一样有自己的程序装载与运行机制，内存管理机制，线程及栈管理机制，看起来就像是一台完整的计算机，这就是 JVM 被称作 machine（机器）的原因。

## JVM 的垃圾回收

事实上，JVM 比操作系统更进一步，它不但可以管理内存，还可以对内存进行自动垃圾回收。所谓自动垃圾回收就是将 JVM 堆中的已经不再被使用的对象清理掉，释放宝贵的内存资源。那么要想进行垃圾回收，首先一个问题就是如何知道哪些对象是不再被使用的，可以清理的呢？

JVM 通过一种可达性分析算法进行垃圾对象的识别，具体过程是：从线程栈帧中的局部变量，或者是方法区的静态变量出发，将这些变量引用的对象进行标记，然后看这些被标记的对象是否引用了其他对象，继续进行标记，所有被标记过的对象都是被使用的对象，而那些没有被标记的对象就是可回收的垃圾对象了。所以你可以看出来，可达性分析算法其实是一个引用标记算法。

进行完标记以后，JVM 就会对垃圾对象占用的内存进行回收，回收主要有三种方法。

**第一种方式是清理：**将垃圾对象占据的内存清理掉，其实 JVM 并不会真的将这些垃圾内存进行清理，而是将这些垃圾对象占用的内存空间标记为空闲，记录在一个空闲列表里，当应用程序需要创建新对象的时候，就从空闲列表中找一段空闲内存分配给这个新对象。

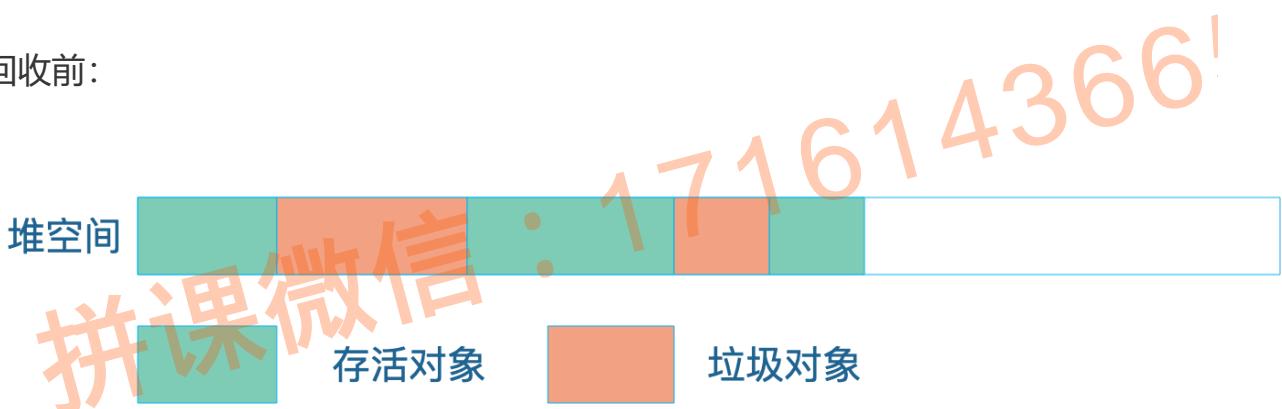
但这样做有一个很明显的缺陷，由于垃圾对象是散落在内存空间各处的，所以标记出来的空闲空间也是不连续的，当应用程序创建一个数组需要申请一段连续的大内存空间时，即使堆空间中有足够的空闲空间，也无法为应用程序分配内存。

**第二种方式是压缩：**从堆空间的头部开始，将存活的对象拷贝放在一段连续的内存空间中，那么其余的空间就是连续的空闲空间。

**第三种方法是复制：**将堆空间分成两部分，只在其中一部分创建对象，当这个部分空间用完的时候，将标记过的可用对象复制到另一个空间中。JVM 将这两个空间分别命名为 from 区域和 to 区域。当对象从 from 区域复制到 to 区域后，两个区域交换名称引用，继续在 from 区域创建对象，直到 from 区域满。

下面这系列图可以让你直观地了解 JVM 三种不同的垃圾回收机制。

回收前：



清理：



压缩：

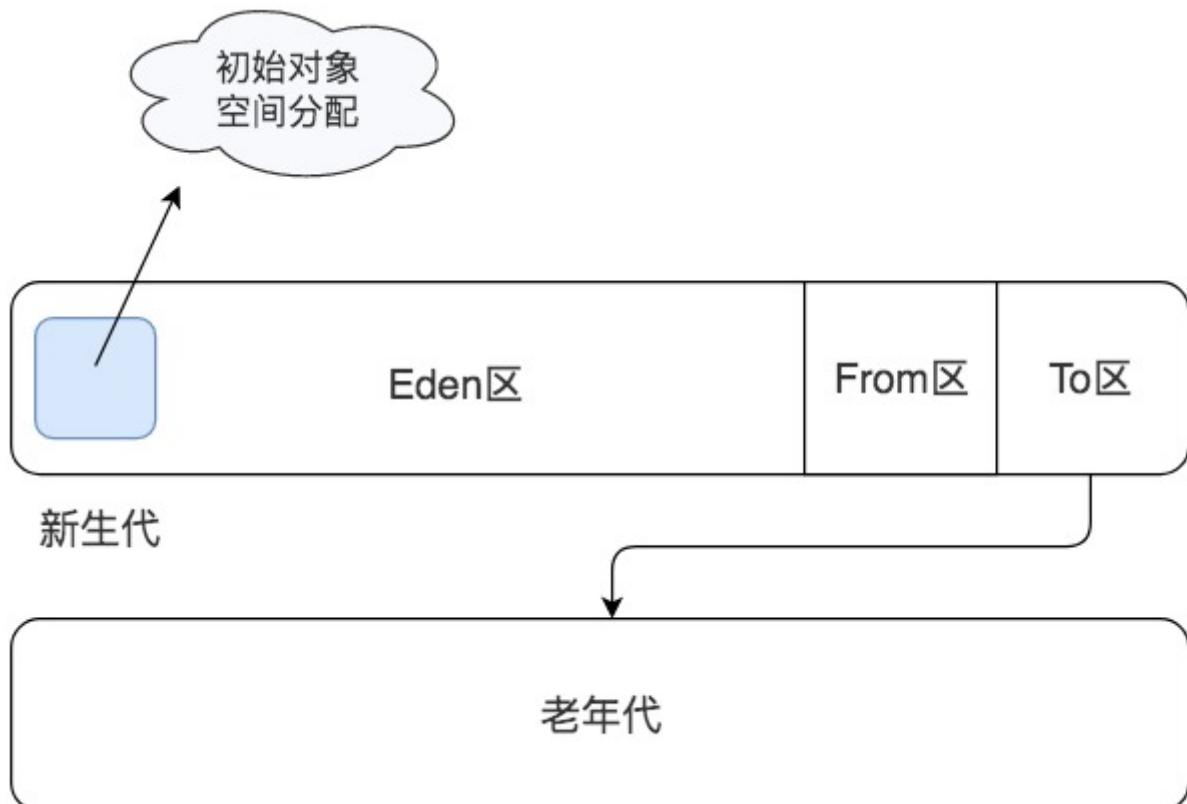


复制：

JVM 在具体进行垃圾回收的时候，会进行分代回收。绝大多数的 Java 对象存活时间都非常短，很多时候就在一个方法内创建对象，对象引用放在栈中，当方法调用结束，栈帧出栈的时候，这个对象就失去引用了，成为垃圾。针对这种情况，JVM 将堆空间分成新生代 (young) 和老年 (old) 两个区域，创建对象的时候，只在新生代创建，当新生代空间不足的时候，只对新生代进行垃圾回收，这样需要处理的内存空间就比较小，垃圾回收速度就比较快。

新生代又分为 Eden 区、From 区和 To 区三个区域，每次垃圾回收都是扫描 Eden 区和 From 区，将存活对象复制到 To 区，然后交换 From 区和 To 区的名称引用，下次垃圾回收的时候继续将存活对象从 From 区复制到 To 区。当一个对象经过几次新生代垃圾回收，也就是几次从 From 区复制到 To 区以后，依然存活，那么这个对象就会被复制到老年区。

当老年区空间已满，也就是无法将新生代中多次复制后依然存活的对象复制进去的时候，就会对新生代和老年区的内存空间进行一次全量垃圾回收，即 Full GC。所以根据应用程序的对象存活时间，合理设置老年区和新生代的空间比例对 JVM 垃圾回收的性能有很大影响，JVM 设置老年区新生代比例的参数是 -XX:NewRatio。



JVM 中，具体执行垃圾回收的垃圾回收器有四种。

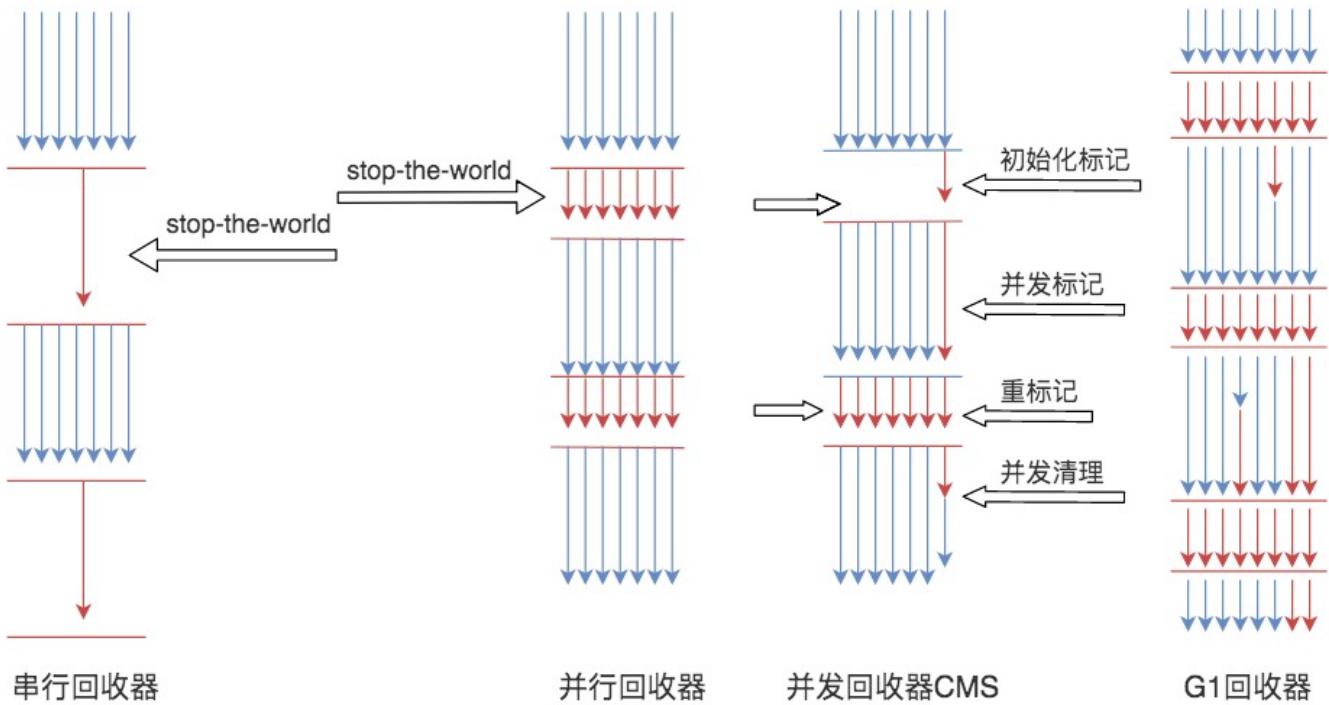
**第一种是 Serial 串行垃圾回收器**，这是 JVM 早期的垃圾回收器，只有一个线程执行垃圾回收。

**第二种是 Parallel 并行垃圾回收器**，它启动多线程执行垃圾回收。如果 JVM 运行在多核 CPU 上，那么显然并行垃圾回收要比串行垃圾回收效率高。

在串行和并行垃圾回收过程中，当垃圾回收线程工作的时候，必须要停止用户线程的工作，否则可能会导致对象的引用标记错乱，因此垃圾回收过程也被称为 stop the world，在用户视角看来，所有的程序都不再执行，整个世界都停止了。

**第三种 CMS 并发垃圾回收器**，在垃圾回收的某些阶段，垃圾回收线程和用户线程可以并发运行，因此对用户线程的影响较小。Web 应用这类对用户响应时间比较敏感的场景，适用 CMS 垃圾回收器。

**最后一种是 G1 垃圾回收器**，它将整个堆空间分成多个子区域，然后在这些子区域上各自独立进行垃圾回收，在回收过程中垃圾回收线程和用户线程也是并发运行。G1 综合了以前几种垃圾回收器的优势，适用于各种场景，是未来主要的垃圾回收器。



小结

我们为什么需要了解 JVM 呢？JVM 有很多配置参数，Java 开发过程中也可能会遇到各种问题，了解了 JVM 的基本构造，就可以帮助我们从原理上去解决问题。

比如遇到 `OutOfMemoryError`, 我们就知道是堆空间不足了, 可能是 JVM 分配的内存空间不足以让程序正常运行, 这时候我们需要通过调整 `-Xmx` 参数增加内存空间。也可能是程序存在内存泄漏, 比如一些对象被放入 `List` 或者 `Map` 等容器对象中, 虽然这些对象程序已经不再使用了, 但是这些对象依然被容器对象引用, 无法进行垃圾回收, 导致内存溢出, 这时候可以通过 `jmap` 命令查看堆中的对象情况, 分析是否有内存泄漏。

如果遇到 `StackOverflowError`, 我们就知道是线程栈空间不足, 栈空间不足通常是因为方法调用的层次太多, 导致栈帧太多。我们可以先通过栈异常信息观察是否存在错误的递归调用, 因为每次递归都会使嵌套方法调用更深入一层。如果调用是正常的, 可以尝试调整 `-Xss` 参数增加栈空间大小。

如果程序运行卡顿，部分请响应延迟比较厉害，那么可以通过 `jstat` 命令查看垃圾回收器的运行状况，是否存在较长时间的 FullGC，然后调整垃圾回收器的相关参数，使垃圾回收对程序运行的影响尽可能小。

执行引擎在执行字节码指令的时候，是解释执行的，也就是每个字节码指令都会被解释成一个底层的 CPU 指令，但是这样的解释执行效率比较差，JVM 对此进行了优化，将频繁执行的代码编译为底层 CPU 指令存储起来，后面再执行的时候，直接执行编译好的指令，不再解释执行，这就是 JVM 的即时编译 JIT。Web 应用程序通常是长时间运行的，使用 JIT 会有很好的优化效果，可以通过 `-server` 参数打开 JIT 的 C2 编译器进行优化。

总之，如果你理解了 JVM 的构造，在进行 Java 开发的时候，遇到各种问题，都可以思考一下，这在 JVM 层面是如何的？然后进一步查找资料、分析问题，这样就会真正解决问题，而且经过这样不断地思考分析，你对 Java，对 JVM，甚至对整个计算机的原理体系以及设计理念都会有更多认识和领悟。

## 思考题

你在 Java 开发过程中遇到过什么样的问题？这些问题和 JVM 底层原理是怎样的关系？

你有想过这些问题吗？你可以把你的疑惑或者想法写在评论区里，集思广益。也欢迎你把这篇文章分享给你的朋友或者同事，一起交流一下。

扫码参加 21 天打卡计划

# 搞定后端技术基础



扫一扫参与小程序话题



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金奖励**。

## 精选留言 (12)

 写留言



幸福来敲门

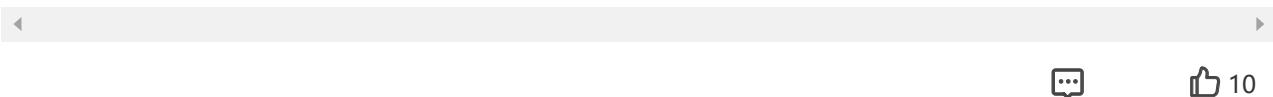
2019-11-22

老师，内存溢出和内存泄漏有什么区别，以java为例？？

作者回复: 内存溢出是说程序需要申请的内存超过了JVM当前可以分配的最大内存，溢出。

内存泄漏是说期望被回收的内存对象没有被回收，泄漏。

内存泄漏持续发生，很可能引起内存溢出。



[耶]友超

2019-11-22

老师，好强大的内功

展开 ▾



Solomon

2019-11-23

老师你好

1.为什么新生代要设置3个区域呢？只有from和to不行吗？

2.多线程执行的时候，栈帧在栈上如何分配空间的，是分多个java栈吗？

展开 ▾



y欧尼酱

2019-11-22

老师能多讲讲类加载吗，这儿只知道jvm加载class字节码，在深入就不懂了，谢谢老师。



乘坐Tornado的线程魔...

2019-11-22

存活对象从From区复制到To区之后，在交换名称之前，是不是要把From区的那些不存活

对象全部回收？（保证交换名字之后，To区永远是空白的）

展开 ▾



乘坐Tornado的线程魔...

2019-11-22

老师好，有关JVM运行前的编译过程，请问HotSpot编译器和IDEA编译器的区别是什么？



KL3

2019-11-22

请问老师，关于类加载，对象方法和类方法都是放在方法区吗？



老男孩

2019-11-22

虽然之前学习过相关的知识，老师讲的不敢说是最全面的，但是非常易于理解，我对jvm相关知识又有了新的认识！受益匪浅！

1) 如果能把类加载机制说的更详细一点就更好了。因为以前我遇到过一个问题，就是tomcat启动后无法访问，一看日志报了一个PermGen空间不足的error，后来重新设置了一下就好了。原因可能是加了很多依赖jar包导致PermGen空间不足了。当时没有再多想。加...  
展开 ▾



我爱布丁

2019-11-22

老师，第一讲中谈到当操作系统载入编译后程序时，会在内存中建立进程，划分代码段，堆，栈等。那么当操作系统运行JAVA虚拟机加载类文件时，在虚拟机中的Java堆，栈是否会直接map到JAVA虚拟机进程的堆，栈呢？从操作系统看一个JAVA虚拟机进程，和一个普通的编译后程序进程结构上有区别吗？

展开 ▾

作者回复：虚拟机内部的堆和栈不会map到操作系统的堆和栈，这样才能跨平台。

操作系统看JVM就是一个普通进程。



miracle

2019-11-23

虽然这个对象被放置在堆中，但是这个对象不会被其他线程访问到，也是线程安全的。

请问下，多个线程都可以访问这个方法，进而可以操作堆中的对象，为什么是线程安全的

展开 ▾

💬 2



## 乘坐Tornado的线程魔...

2019-11-22

操作系统处理线程栈这个场景是不是可以想象成，高并发场景下，每一个请求都要压入操作系统的线程栈，请求返回后，执行这个请求的线程才能从操作系统的线程栈中出栈？

💬



尹宗昌

2019-11-22

老师，之前看到，MySQL connector-j 驱动在特定模式下 (useCompression=true) 的内存泄漏问题。能简单提个思路，这样也能循序渐进

展开 ▾

💬

