

理解 JVM 笔记系列 (二) : 内存模型 (附脑图)

作者: [hudk](#)

原文链接: <https://ld246.com/article/1590577990315>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

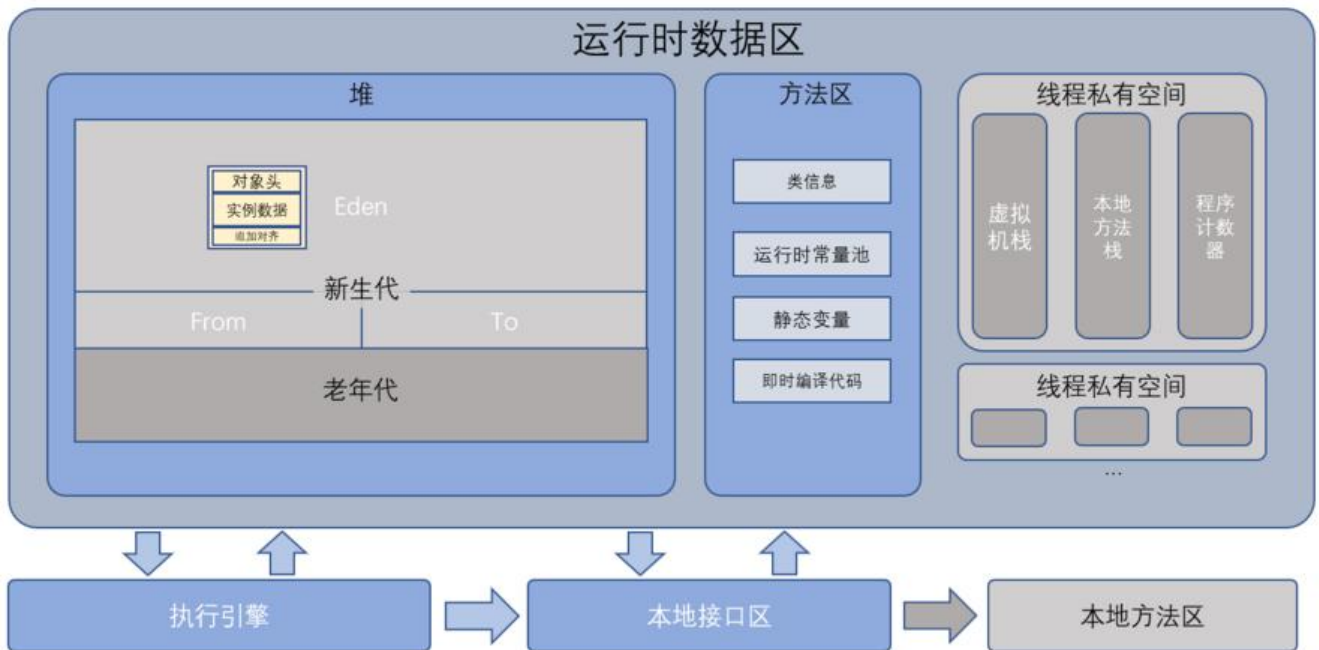
概述

由于Java程序运行过程中，对内存的管理和控制，全权交给了Java虚拟机来管理，虽然可以避免绝大多数的内存溢出问题，但也不是绝对的，特殊的代码编写也会让程序出现一定程度上的内存溢出情况，且出现这种问题，程序员不了解虚拟机的内存模型和管理方式，要找寻和解决问题，将会异常的困难所以需要了解Java虚拟机的内存模型。

总体上虚拟机的运行时数据区域结构分为线程私有和公共区域：

- 线程私有
 - 程序计数器
 - 虚拟机栈
 - 本地方法栈
- 公共区域
 - 堆
 - 方法区

以上全部是逻辑上的概念，具体的虚拟机实现，在物理内存如何映射，这里先不做考虑。主要是认识他们在概念上是什么含义，以及在虚拟机规范中，他们分别应该承担什么样的“职责”。



程序计数器

这是一个在虚拟机中空间占用较小的一块区域，它专门用来保存当前线程执行的下一条字节码指令“号”，这样程序运行时，才知道下一条指令应该去哪里找到并运行。当然从多线程的角度考虑，由于序计数器是线程私有的，在线程切换过程中，比如线程等待或堵塞被唤醒后，可以依据程序计数器中内容，继续执行刚才将要执行的指令。当正在执行本地方法时，这个计数器的值应该为空；另外，此存区域是规范中唯一没有规定OutOfMemoryError情况的区域。

Java虚拟机栈、

在线程运行到每一个新方法时，虚拟机栈都会创建一个新的栈帧作为栈顶，每一个栈帧中，包含了当执行方法的局部变量，操作数栈，方法引用（动态链接），方法的出口这些信息，当前方法运行结束，该栈帧随之被弹出（删除）。

栈帧结构图：



本地方法栈

与虚拟机栈很类似，不同的是，本地方法栈是提供虚拟机调用本地方法时专门为运行本地方法而准备内存区域。

Java 堆

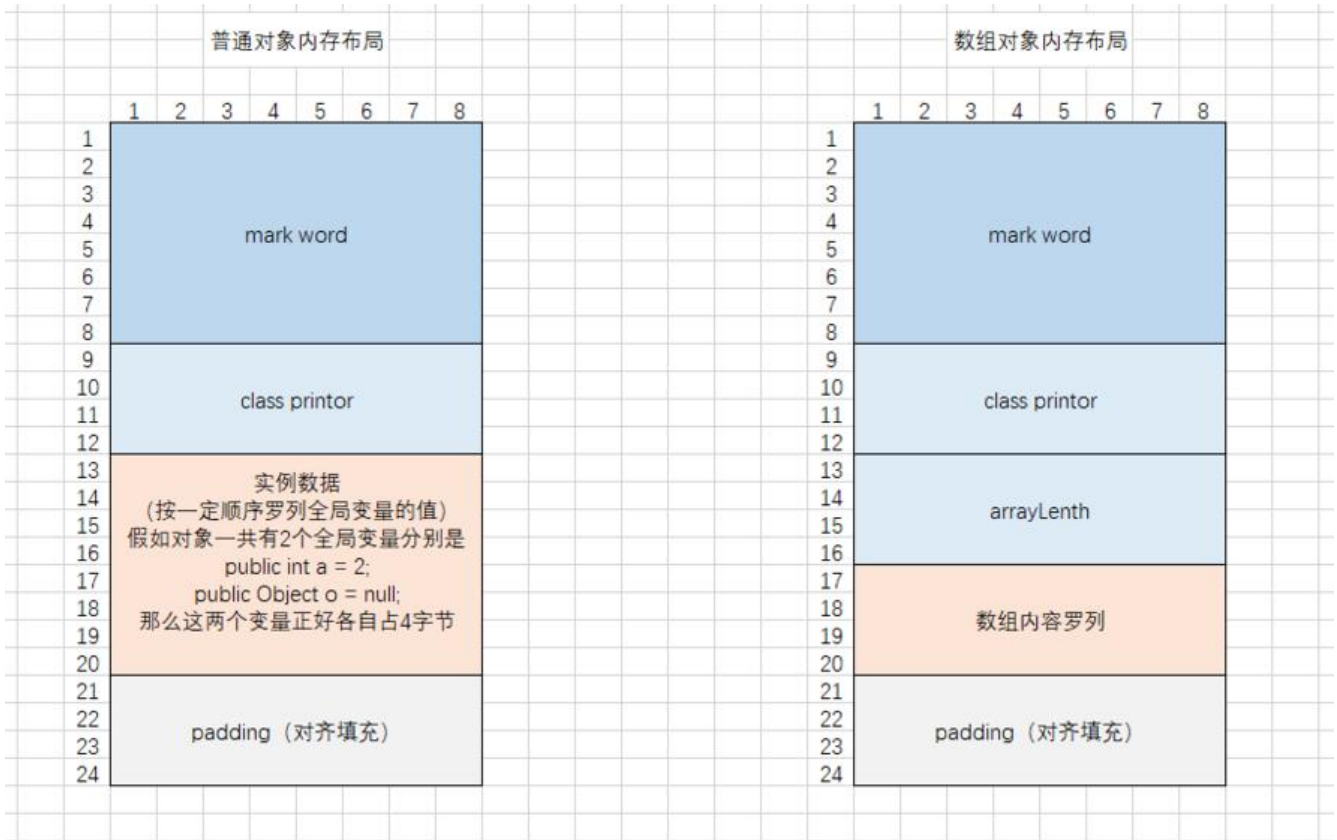
这是Java 虚拟机内存管理的最大的一块区域，这块区域的主要职责是存放Java世界中几乎所有的对象例数据，是虚拟机中所有线程可以共同享有访问权力的内存区域。并且这块内存区域是虚拟机垃圾收器管理的区域，也被成为“GC堆”。从垃圾收集的角度来考虑，Java堆有可以分为以下更细粒度的区域。

- 新生代
 - Eden
 - From Survivor
 - To Survivor

- 老年代
- 永久代

这样来划分堆，单纯是为了垃圾回收算法来服务的，换句话说，高效的垃圾回收算法，决定了堆需要何种方式来存储对象实例。从数据结构角度考虑，这样划分堆，本质上就是为垃圾回收算法提供更高的数据结构。

对象在堆中的结构：



方法区

方法区也是虚拟机中的公共存储区域，主要保存了被虚拟机加载的类型信息，常量，静态变量，即时译后的代码缓存等。

运行时常量池

这是属于方法区的一部分，主要用来保存类型信息中的常量信息。并且支持将运行时产生的新常量加其中。

根据《深入理解Java虚拟机》第3版相关章节总结整理的脑图

运行时数据区域

程序计数器

- 是什么** 可以看作是当下线程执行的字节码指令行的指示器
- 作用** 通过改变程序计数器，来找到下一条要执行的指令是什么，反映到程序执行过程的顺序执行、分支执行、循环执行、跳转执行、异常处理、线程恢复等过程。
- 特点**
 - 属于线程私有的
 - 执行本地方法时，该计数器为空 (undefined)
 - 唯一一个在《Java虚拟机规范》中没有规定任何OutOfMemoryError情况的区域

虚拟机栈

- 是什么** 反映了Java程序执行方法时方法之间的调用关系的内存模型
- 栈帧**
 - 对应每个方法，栈帧中存储了对应方法的相关数据，包括：存储局部变量表、操作数栈、动态链接、方法出口
 - 局部变量表**
 - 保存了方法参数列表变量，方法内定义的局部变量
 - 变量类型**
 - 基本数据类型
 - 对象引用
 - returnAddress类型
 - 局部变量槽 (slot)**
 - 其中64位长度的long和double类型的数据会占用两个变量槽，其余的数据类型只占用一个
 - 虚拟机真正使用多大的内存空间 (假如按照1个变量槽占用32个比特，64个比特，或者更多) 来实现一个变量槽，这是完全由具体的虚拟机实现自行决定的事情
 - 操作数栈** 方法内指令执行时，用于存放指令的操作数
 - 动态链接**
 - 方法出口**
- 异常**
 - StackOverflowError** 如果线程请求的栈深度大于虚拟机所允许的深度
 - OutOfMemoryError**
 - 如果Java虚拟机栈容量可以动态扩展，当栈扩展时无法申请到足够的内存
 - HotSpot虚拟机的栈容量是不可以动态扩展的，以前的Classic虚拟机是可以，所以在HotSpot虚拟机上不会由于虚拟机栈无法扩展而导致OutOfMemoryError异常，只要线程申请栈空间成功就不会有OOM，但是如果申请时就失败，仍然是会出现OOM异常的。

本地方法栈

本地方法栈 (Native Method Stacks) 与虚拟机栈所发挥的作用是非常相似的，其区别只是虚拟机栈为虚拟机执行Java方法 (也就是字节码) 服务，而本地方法栈则是为虚拟机使用到的本地 (Native) 方法服务。

堆

- 是什么** Java虚拟机中占用空间最大的一块运行时存储区域，用于存放和管理运行时创建的对象实例数据
- 特点**
 - 线程共享
 - 绝大部分对象实例存在这里，但不是全部
- 分区**
 - 分区的目的是为了理解和服务对应垃圾收集工作，换句话说，堆的唯一目的就是存放对象实例数据，以供程序使用，但为了管理运行时对象实例的动态变化造成的内存不够和碎片问题，不得不想办法将堆分成一些逻辑分区，一是满足对应垃圾收集算法，二是提高虚拟机性能和内存使用效率，基于这种情况，不同的虚拟机对堆的划分会有所不同，但大都会将堆进行分代划分成年轻代、老年代、永久代 (或元空间) 等概念。
- 垃圾收集算法**
 - 标记死亡**
 - 引用计数法
 - 可达性分析
 - 清除回收**
 - 标记-清除算法
 - 标记-复制算法
 - 标记-整理算法
- 异常** OutOfMemoryError 如果在Java堆中没有内存完成实例分配，并且堆也无法再扩展时

方法区

- 是什么** 各个线程共享的内存区域，它用于存储已被虚拟机加载的类型信息、常量、静态变量、即时编译器编译后的代码缓存等数据
- 特点** JDK1.8以后，方法区的实现依赖于本地内存的元空间
- 运行时常量池**
 - Class文件中常量池表 (Constant Pool Table)，在未加载后存储到方法区的运行时常量池中。
 - 动态性：运行期间也可以将新的常量放入池中
- 异常** OutOfMemoryError

直接内存

- 不是虚拟机运行时数据区的一部分，也不是《Java虚拟机规范》中定义的内存区域
- 使用Native函数库直接分配堆外内存
- DirectByteBuffer对象作为这块内存的引用进行操作
- 会受到物理内存大小的限制
- 物理内存也无法满足时将出现OutOfMemoryError异常