# JVM 内存模型

HotSpot 内存模型 -- JDK8

image

image

* **Heap**: Java 堆是可供各线程共享的运行时内存区域，是 Java 虚拟机所管理的内存区域中最大的一块。此区域非常重要，几乎所有的对象实例和数组实例都要在 Java 堆上分配，但随着 JIT 编译器及逃逸分析技术的发展，**也可能会被优化为栈上分配**
* **Internd String**: 字符串字面量常量池
* **Calss Meta Data**: 每一个类的结构信息，比如 字段 和 方法数据、构造函数和普通方法的字节码内容，还包括一些初始化的时候用到的特殊方法。
* **Runtime Constant Pool**: 运行时常量池是 class 文件中每一个类或接口的 常量池表(Constant Pool)的运行时表示形式，是方法区的一部分。它包括了若干种不同的常量。常量池表存放编译器生成的 **各种字面量和符号引用**，这部分内容将在类加载后进入方法区的运行时常量池中存放。运行时常量池具有动态性，运行期间也可以将新的量放到运行时常量池中，典型的应用是 String 类的 intern 方法
* **Code Cache**: JIT 编译缓存的本地代码
* **PC Register**: CPU内部的寄存器中就包含一个程序计数器，存放程序执行的下一条指令地址。
* **JVM Stacks**: Java 虚拟机栈也是线程私有的，每一条线程都拥有自己私有的Java 虚拟机栈，它与线程同时创建。它描述了 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，用于存储局部变量表、操作数栈、动态链接、方法出口等信息。
* **Native Stacks**: 本地方法栈和 Java 虚拟机栈的作用相似，Java 虚拟机栈执行的是字节码，而本地方法栈执行的是 native 方法。本地方法栈使用传统的栈（C Stack）来支持 native 方法。

JDK 1.7开始，字符串常量和符号引用等就被移出永久代： - 符号引用迁移至系统堆内存(Native Heap) - 字符串字面量迁移至Java堆(Java Heap)

在 JVM 规范中，并不存在这么多分区，只包含 5 大分区：

image

image

* Metahod Area
* Heap
* JVM Stack
* PC Register
* Native Stack

在 HotSpot 中，方法区涵盖了除 Heap,JVM Stack,PC Register,Native Stack 之外的其他区域。

## 参考链接

* [JEP 122: Remove the Permanent Generation](http://openjdk.java.net/jeps/122)
* [Java JVM Run-time Data Areas - Javapapers](https://javapapers.com/core-java/java-jvm-run-time-data-areas/#Run_time_Constant_Pool)
* [JVM Internals](http://blog.jamesdbloom.com/JVMInternals.html#constant_pool)
* [深入探究 JVM | Java 的内存区域解析](https://www.sczyh30.com/posts/Java/jvm-memory/)
* [The Java Virtual Machine](http://www.artima.com/insidejvm/ed2/jvm2.html)
* [The Java HotSpot Performance Engine Architecture](https://www.oracle.com/technetwork/java/whitepaper-135217.html)