对于Java程序员来说，在虚拟机自动内存管理机制的帮助下，不再需要为每一个new操作去写配对

的delete/free代码，不容易出现内存泄漏和内存溢出问题，

在《Java虚拟机规范》中，对这个内存区域规定了两类异常状况：如果线程请求的栈深度大于虚

拟机所允许的深度，将抛出StackOverflowError异常；如果Java虚拟机栈容量可以动态扩展[2]，当栈扩

展时无法申请到足够的内存会抛出OutOfMemoryError异常。

native方法称为本地方法。在java源程序中以关键字“native”声明，不提供函数体。  
其实现使用C/C++语言在另外的文件中编写，编写的规则遵循Java本地接口的规范(简称JNI)。  
简而言就是Java中声明的可调用的使用C/C++实现的方法。

所有

的对象实例以及数组都应当在堆上分配[1]”

Java对象实例都分配在堆上也渐渐变得不是那么绝对了

[1] 《Java虚拟机规范》中的原文：The heap is the runtime data area from which memory for all class

instances and arrays is allocated。

程序计数器（Program Counter Register）是一块较小的内存空间，它可以看作是当前线程所执行的

字节码的行号指示器。在Java虚拟机的概念模型里[1]，

虚拟机栈描述的是Java方法执行的线程内存模型：每个方法被执行的时候，Java虚拟机都

会同步创建一个栈帧[1]（Stack Frame）用于存储局部变量表、操作数栈、动态连接、方法出口等信

息。每

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。

对于Java应用程序来说，Java堆（Java Heap）是虚拟机所管理的内存中最大的一块。Java堆是被所

有线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对象实例，Java

世界里“几乎”所有的对象实例都在这里分配内存。在《Java虚拟机规范》中对Java堆的描述是：“所有

的对象实例以及数组都应当在堆上分配[1]”，

方法区（Method Area）与Java堆一样，是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。

运行时常量池（Runtime Constant Pool）是方法区的一部分。Class文件中除了有类的版本、字

段、方法、接口等描述信息外，还有一项信息是常量池表（Constant Pool Table），用于存放编译期生

成的各种字面量与符号引用，这部分内容将在类加载后存放到方法区的运行时常量池中。

当Java虚拟机遇到一条字节码new指令时，首先将去检查这个指令的参数是否能在常量池中定位到

一个类的符号引用，并且检查这个符号引用代表的类是否已被加载、解析和初始化过。如果没有，那

必须先执行相应的类加载过程

为新生对象分配内存

Java堆内存的OutOfMemoryError异常是实际应用中最常见的内存溢出异常情况。出现Java堆内存

溢出时，异常堆栈信息“java.lang.OutOfMemoryError”会跟随进一步提示“Java heap space”。