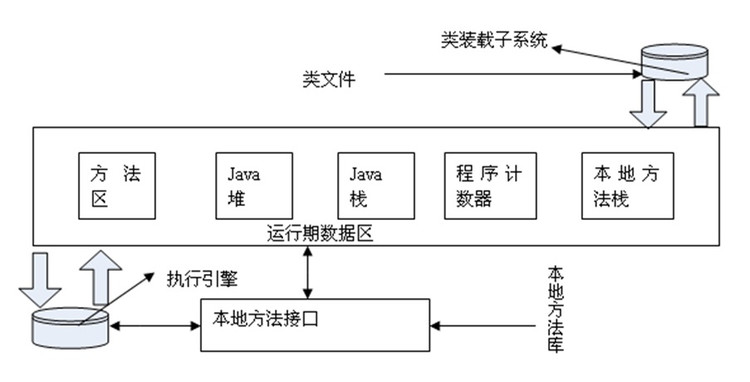
JVM内存布局：主要分为4个部分，堆，栈，方法区，程序计数器。



* 堆(Heap)：对象的实例都是建立在堆上。该区域会频繁的产生新的对象和回收死亡的对象，所以是内存回收的主要区域。该部分可能出现Out Of Memory。
* 栈(Stack)：分为虚拟机栈和本地方法栈。虽然sun 的Hotspot没有这么区分。该部分可能出现Out Of Memory。
* 虚拟机栈(JVM Stack)：用于存储局部变量表，操作栈，动态链接，方法出口等信息。
* 局部变量表：存放编译时可知的各种基本类型(boolean, byte, char, short , int, float, long, double),对象引用(reference, 对象指针的地址)和return address. 局部变量表的内存大小在编译时就已经确定。在运行方法时，不会改变局部变量表的大小。
* 本地方法栈(Native Stack)：和虚拟机方法栈类似，唯一的区别就是本地方法栈是在调用本地native方法的时候使用。
* 方法区(Method Area)：存的是类的信息，常量，静态变量，即时编译器编译后的代码等数据。该部分也可能出现Out Of Memory，需要不断的动态创建对象。这个区域也曾经被叫做永久代，也就是说不会发生内存回收。但是现在该区域也已经被纳入到内存回收的队列，不再是永久代了。
* 运行时常量池(Run time Constants Pool)：用于存放编译时产生的字面常量和符号引用。
* 程序计数器(Program Counter Register)：JVM使用的是线程轮转的方式执行线程，所以需要程序计数器来保存线程的执行的栈帧。该区域是用户无法触及到的，所以不会出现Out Of Memory.
* 直接内存(Directly Memory): 这部分并不是JVM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。在JDK1.4中加入了NIO(new Input/Output), 引入了一种基于通道和缓冲区的I/O方式。它可以直接使用Native函数去分配堆外内存，然后直接通过一个存在java堆里DirectByteBuffer对象作为这块内存的引用进行操作。这样在一些场景中可以显著提高性能，因为避免了java堆和native堆之间的来回拷贝。