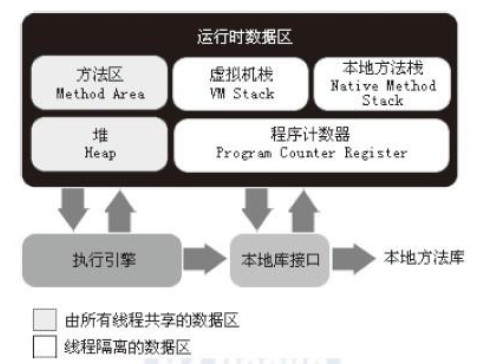
**1、介绍**

  JAVA虚拟机在执行Java程序的过程中会把他所管理的内存划分为若干个不同的数据区域。这些区域都有各自的用途，以及创建和销毁的时间，有的区域随着虚拟机进程的启动而存在，有些区域则依赖用户线程的启动和结束而建立和销毁。

**1.1 内存区域**

  Java虚拟机在执行Java程序的过程中会把他所管理的内存划分为若干个不同的数据区域。Java虚拟机规范将JVM所管理的内存分为以下几个运行时数据区：程序计数器、Java虚拟机栈、本地方法栈、Java堆、方法区。整体划分情况如下图所示：



其中,方法区和和堆区是线程共享的，其它区域是线程隔离的。

**1.1.1 程序计数器**

  一块较小的内存空间，它**是当前线程所执行的字节码的行号指示器**，字节码解释器工作时**通过改变该计数器的值来选择下一条需要执行的字节码指令**，分支、跳转、循环等基础功能都要依赖它来实现。每条线程都有一个独立的的程序计数器，各线程间的计数器互不影响，因此该区域是线程私有的。

  当线程在执行一个Java方法时，该计数器记录的是正在执行的虚拟机字节码指令的地址，当线程在执行的是Native方法（调用本地[操作系统](http://lib.csdn.net/base/operatingsystem)方法）时，该计数器的值为空。另外，该内存区域是唯一一个在Java虚拟机规范中么有规定任何OOM（内存溢出：OutOfMemoryError）情况的区域。

**1.1.2 Java虚拟机栈**

 在Java虚拟机规范中，对这个区域规定了两种异常情况：

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**。

如果虚拟机在**动态扩展栈时**无法申请到足够的内存空间，则抛出**OutOfMemoryError异常**。

这两种情况存在着一些互相重叠的地方：当栈空间无法继续分配时，到底是内存太小，还是已使用的栈空间太大，其本质上只是对同一件事情的两种描述而已。在单线程的操作中，无论是由于栈帧太大，还是虚拟机栈空间太小，当栈空间无法分配时，虚拟机抛出的都是StackOverflowError异常，而不会得到OutOfMemoryError异常。而在多线程环境下，则会抛出OutOfMemoryError异常。

下面详细说明栈帧中所存放的各部分信息的作用和数据结构。

**（1）局部变量表**

     局部变量表是一组变量值存储空间，**用于存放方法参数和方法内部定义的局部变量，其中存放的数据的类型是编译期可知的各种基本数据类型、对象引用（reference）和returnAddress类型（它指向了一条字节码指令的地址）。**局部变量表所需的内存空间在编译期间完成分配，即在Java程序被编译成Class文件时，就确定了所需分配的最大局部变量表的容量。当进入一个方法时，这个方法需要在栈中分配多大的局部变量空间是完全确定的，在方法运行期间不会改变局部变量表的大小。

局部变量表的容量以变量槽（Slot）为最小单位。在虚拟机规范中并没有明确指明一个Slot应占用的内存空间大小（允许其随着处理器、操作系统或虚拟机的不同而发生变化），一个Slot可以存放一个32位以内的数据类型：boolean、byte、char、short、int、float、reference和returnAddresss。reference是对象的引用类型，returnAddress是为字节指令服务的，它执行了一条字节码指令的地址。对于64位的数据类型（long和double），虚拟机会以高位在前的方式为其分配两个连续的Slot空间。

虚拟机通过索引定位的方式使用局部变量表，索引值的范围是从0开始到局部变量表最大的Slot数量，对于32位数据类型的变量，索引n代表第n个Slot，对于64位的，索引n代表第n和第n+1两个Slot。

在方法执行时，虚拟机是使用局部变量表来完成参数值到参数变量列表的传递过程的，如果是实例方法（非static），则局部变量表中的第0位索引的Slot默认是用于传递方法所属对象实例的引用，在方法中可以通过关键字“this”来访问这个隐含的参数。其余参数则按照参数表的顺序来排列，占用从1开始的局部变量Slot，参数表分配完毕后，再根据方法体内部定义的变量顺序和作用域分配其余的Slot。

局部变量表中的Slot是可重用的，方法体中定义的变量，作用域并不一定会覆盖整个方法体，如果当前字节码PC计数器的值已经超过了某个变量的作用域，那么这个变量对应的Slot就可以交给其他变量使用。这样的设计不仅仅是为了节省空间，在某些情况下Slot的复用会直接影响到系统的而垃圾收集行为。

**（2）操作数栈**

     操作数栈又常被称为操作栈，操作数栈的最大深度也是在编译的时候就确定了。32位数据类型所占的栈容量为1,64为数据类型所占的栈容量为2。当一个方法开始执行时，它的操作栈是空的，在方法的执行过程中，会有各种字节码指令（比如：**加操作、赋值运算等**）向操作栈中写入和提取内容，也就是入栈和出栈操作。

Java虚拟机的解释执行引擎称为“基于栈的执行引擎”，其中所指的“栈”就是操作数栈。因此我们也称Java虚拟机是基于栈的，这点不同于[Android](http://lib.csdn.net/base/android)虚拟机，Android虚拟机是基于寄存器的。

基于栈的指令集最主要的优点是可移植性强，主要的缺点是执行速度相对会慢些；而由于寄存器由硬件直接提供，所以基于寄存器指令集最主要的优点是执行速度快，主要的缺点是可移植性差。

**（3）动态连接**

**每个栈帧都包含一个指向运行时常量池（在方法区中，后面介绍）中该栈帧所属方法的引用**，**持有这个引用是为了支持方法调用过程中的动态连接。**Class文件的常量池中存在有大量的符号引用，字节码中的方法调用指令就以常量池中指向方法的符号引用为参数。这些符号引用，一部分会在类加载阶段或第一次使用的时候转化为直接引用（如final、static域等），称为静态解析，另一部分将在每一次的运行期间转化为直接引用，这部分称为动态连接。

**（4）方法返回地址**

     当一个方法被执行后，有两种方式退出该方法：执行引擎遇到了任意一个方法返回的字节码指令或遇到了异常，并且该异常没有在方法体内得到处理。无论采用何种退出方式，在方法退出之后，都需要返回到方法被调用的位置，程序才能继续执行。方法返回时可能需要在栈帧中保存一些信息，用来帮助恢复它的上层方法的执行状态。一般来说，方法正常退出时，调用者的PC计数器的值就可以作为返回地址，栈帧中很可能保存了这个计数器值，而方法异常退出时，返回地址是要通过异常处理器来确定的，栈帧中一般不会保存这部分信息。

方法退出的过程实际上等同于把当前栈帧出站，因此退出时可能执行的操作有：恢复上层方法的局部变量表和操作数栈，如果有返回值，则把它压入调用者栈帧的操作数栈中，调整PC计数器的值以指向方法调用指令后面的一条指令。

**1.1.3 本地方法栈**

该区域与虚拟机栈所发挥的作用非常相似，只是虚拟机栈为虚拟机执行Java方法服务，而本地方法栈则为使用到的**本地操作系统（Native）方法服务**。在虚拟机规范中对本地方法栈中方法使用的语言，使用方式与数据结构并没有强制规定，因此虚拟机可以自由实现。有些虚拟机比如（Sun HotSpot虚拟机）直接将本地方法栈和虚拟机栈合二为一。与虚拟机栈一样，本地方法区域也会抛出StackOverFlowError(SOF)和OutOfMemoryError(OOM)异常。

**1.1.4 Java堆**

     Java堆（[Java](http://lib.csdn.net/base/java) Heap）是Java虚拟机所管理的内存中最大的一块，它是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，**几乎所有的对象实例和数组**都在这类分配内存。Java Heap是垃圾收集器管理的主要区域，因此很多时候也被称为“**GC堆**”。

根据Java虚拟机规范的规定，Java堆可以处在物理上不连续的内存空间中，只要逻辑上是连续的即可。如果在堆中没有内存可分配时，并且堆也无法扩展时，将会抛出OutOfMemoryError异常。

**1.1.5 方法区（Method Area）**

     方法区也是各个线程共享的内存区域，它**用于存储已经被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据**。**方法区域又被称为“永久代”**，但这仅仅对于Sun HotSpot来讲，JRockit和IBM J9虚拟机中并不存在永久代的概念。Java虚拟机规范把方法区描述为**Java堆的一个逻辑部分**，而且它和Java Heap一样不需要连续的内存，可以选择固定大小或可扩展，另外，虚拟机规范允许该区域可以选择不实现垃圾回收。相对而言，垃圾收集行为在这个区域比较少出现。该区域的内存回收目标主要针是**对废弃常量的和无用类的回收**。

**1.1.6 运行时常量池**

     注意：运行时常量池是方法区的一部分。**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（Class文件常量池）**，用于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存放到方法区的运行时常量池中。运行时常量池相对于Class文件常量池的另一个重要特征是具备动态性，Java语言并不要求常量一定只能在编译期产生，也就是并非预置入Class文件中的常量池的内容才能进入方法区的运行时常量池，运行期间也可能将新的常量放入池中，这种特性被开发人员利用比较多的是String类的intern()方法。

既然运行时常量池是方法区的一部分，自然受到方法区内存限制，当方法区无法满足内存分配需求时，将抛出OutOfMemoryError异常。

**1.1.7  除了上述几个分区外，特别要说明的还有一个是直接内存。**

     直接内存（Direct Memory）并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，它直接从操作系统中分配，因此不受Java堆大小的限制，但是会受到本机总内存的大小及处理器寻址空间的限制，因此它也可能导致OutOfMemoryError异常出现。在JDK1.4中新引入了NIO机制，它是一种基于通道与缓冲区的新I/O方式，可以直接从操作系统中分配直接内存，即在堆外分配内存，这样能在一些场景中提高性能，因为避免了在Java堆和Native堆中来回复制数据。

**2、总结**

上述概念 看起来比较多，而且不容易理解。其实只要抓住两个核心问题即可。

**2.1 Java内存区域到底划分了哪几个区？**

大体上有五个分区 ，分别是堆，方法区，虚拟机栈，本地方法栈，程序计数器。其中堆区和方法区线程共享，其他区域线程隔离。

**2.2 每个区具体放了什么？**

堆区:存放对象实例，GC主要作用区域。

方法区：存放虚拟机加载的类信息，常量，静态变量，即时编译器编译后的代码等数据。

虚拟机栈：描述Java方法执行时的内存模型，为执行Java方法服务。

本地方法栈：描述本地方法Native执行的内存模型，为执行本地方法服务。

程序计数器：为执行字节码指令服务，通过改变计数器值来选取下条指令。